How do you go on, when in your heart
you begin to understand... there is no going back.
There are some things that time cannot mend.
Some hurts that go too deep, that have taken hold.

F. Baggins (The Lord of the Rings III)

In Memory of my Mother

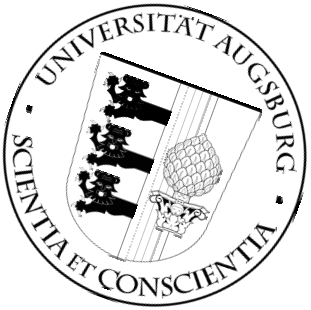
Algebraic Calculi for Hybrid Systems

Referees:
Prof. Dr. Bernhard Möller
Prof. Dr. Gunther Schmidt
Prof. Dr. Walter Vogler

Day of Defense:
July 3, 2009

Algebraic Calculi for Hybrid Systems

Peter Höfner
University of Augsburg



Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

peter@hoefner-online.de

Herstellung und Verlag
Books on Demand GmbH, Norderstedt

ISBN: 978-3-8391-2510-6

Contents

Preamble **1**

1 Introduction **5**
1.1 Motivation 5
1.2 Hybrid Systems 7
1.3 Kleene and Omega Algebras 10
1.4 Contribution and Organisation 13

2 Hybrid Systems — A Short Introduction **15**
2.1 An Introductory Example 15
2.2 Basic Concepts 18
2.3 More Examples 21
2.4 Composing Hybrid Systems 24
2.5 Railway Constructions 29

3 Kleene Algebra and its Extensions **37**
3.1 Kleene Algebras 38
3.2 Relaxing Kleene Algebra 41
3.3 Automation in Kleene Algebra 47
3.4 Basic Properties 49
3.5 Tests 53
3.6 Domain and Codomain 55

4 Algebra of Hybrid Systems **61**
4.1 Trajectory-Based Model 61
4.2 Embedding Hybrid Automata 70
4.3 Composing Hybrid Systems Algebraically 73

4.4 Zeno Effects . . . 77
4.5 Safety and Liveness . . . 85
4.6 Specifications . . . 96

5 Logics for Hybrid Systems **101**
5.1 Logics and Hybrid Systems . . . 101
5.2 Modelling CTL* Algebraically . . . 104
5.3 Modelling Neighbourhood Logic Algebraically . . . 110
5.4 Semiring Neighbours . . . 118
5.5 Applying Algebraic Semantics to Hybrid Systems . . . 124

6 Case Studies **127**
6.1 Algebraic Railway Constructions . . . 127
6.2 Route Planning . . . 130
6.3 An Assembly Line Scheduler . . . 135

7 Conclusion **139**
7.1 Summary . . . 139
7.2 Future Work . . . 140

A Automation in Lazy Kleene Algebras **143**
A.1 Templates . . . 143
A.2 Automated Proofs . . . 146

B Deferred Properties and Proofs **159**
B.1 Additional Properties . . . 159
B.2 Deferred Proofs . . . 161

C Gate Controller Details **175**

Bibliography **183**

List of Figures **200**

List of Figures **201**

Index **202**

Preamble

Übersicht

Fehlerhafte Software hat in den letzten Jahrzehnten zahlreiche Menschenleben gefordert, Umweltschäden hervorgerufen und regelmäßig zu enormen wirtschaftlichen Verlusten geführt. Besonders fehleranfällig sind hierbei Systeme, welche ständig mit ihrer Umwelt interagieren, da sie auf diese flexibel, aber dennoch vorhersagbar reagieren müssen. Anders als für reine Softwaresysteme wie Büroanwendungen, sind Korrektheitsanforderungen in diesen Bereichen besonders hoch — ein Airbag, der sich zu spät öffnet, ist nicht akzeptabel.

Solche sicherheitskritische Systeme können meistens als so genannte *hybride Systeme* charakterisiert werden. Bei diesen Systemen besteht ein Wechselspiel zwischen *kontinuierlichem* Systemverhalten und Kontrollereignissen zu *diskreten* Zeitpunkten, die Zustandswechsel auslösen. Anwendungsgebiete reichen von Steuerungselementen über Medizintechnik bis hin zu Avionik. Aber auch chemische und biologische Systeme können mittels solcher Systeme mathematisch exakt beschrieben werden.

Hybride Systeme sind jedoch häufig so komplex, dass eine computergestützte Verifikation auch mit den heute verfügbaren großen Speicher- und Rechenkapazitäten nicht durchführbar ist. Ziel des vorliegenden Buches ist es Untersuchungen zu einer kompakteren Behandlungsmöglichkeit von Verifikationsaufgaben anzustellen. Zentrales Interesse finden hierbei algebraische Techniken, in denen Systeme durch Gleichungsregeln — ähnlich den aus der Schulalgebra bekannten — beschrieben werden. Die generellen Vorteile eines algebraischen Ansatzes sind vor allem Klarheit und Einfachheit, insbesondere im Hinblick auf (computerunterstützbare) Rechenregeln. Die entwickelte algebraische Charakterisierung hybrider Systeme ermöglicht es beispielsweise, Sicherheitsaspekte mittels einfacher algebraischer Umformungen zu überprüfen. Ferner bietet dieser Ansatz den Vorteil, dass Standard-Computer-Algebrasysteme verwendet werden können. So können *Theorembeweiser* eingesetzt werden, um fundamentale

Eigenschaften hybrider Systeme automatisch zu verifizieren.

Im letzten Jahrzehnt wurden Dutzende unterschiedlicher Logiken für hybride Systeme eingesetzt: Angefangen von klassischer Aussagenlogik über modale und temporale Logiken bis hin zu eigens für diese Systeme entwickelten Logiken. Die meisten dieser Logiken sind für sich wohlverstanden. Aber auf Grund der Vielzahl von verwendeten Begriffe sowie ihrer unterschiedlichen Notation und Bedeutung ist eine uniforme Behandlung der Logiken und ihrer Beziehungen zueinander sehr schwierig. In dem vorliegenden Buch werden daher Untersuchungen zu einer kompakteren und einheitlichen Behandlung angestellt. Zentrales Interesse finden hierbei dieselben algebraischen Techniken wie zur Beschreibung von hybriden Systemen. Durch Algebraisierung ist es möglich, Beziehungen zwischen Logiken aufzuzeigen und, für dieses Buch von besonderer Bedeutung, diese Logiken in einer einheitlichen und systematischen Weise auf hybride Systeme anzuwenden.

Die Forschungsergebnisse umfassen grundlegende Methoden zur Analyse hybrider Systeme und münden in eine kohärente Familie *algebraischer Kalküle für hybride Systeme*. Die Anwendbarkeit und Relevanz der Theorie ist durch erste Fallstudien belegt.

Abstract

Over the past decades incorrect software has claimed numerous lives and has caused environmental damages. Furthermore incorrect software regularly yields economic losses. Systems that interact with their environment are especially fault-prone. On the one hand, they have to be flexible enough to react to environment changes, on the other hand they have to be predictable. Unlike software systems for office work, interacting systems must satisfy restrictive safety criteria — an airbag that inflates too late is unacceptable.

Hybrid systems are often used to model such safety critical systems. They are heterogeneous systems characterised by the interaction of *continuous* dynamics and *discrete* events that cause state changes. They have found widespread applications ranging from control and medical engineering to avionics. But even biological and chemical processes can concisely be described by such systems.

In many cases, hybrid systems are too complex for computer-aided verification — even with today's computers that offer enormous memory and calculating capacities. In this book, we aim at a compact treatment of verification tasks. Algebraic techniques are of particular interest and arise in a natural way. Systems are algebraically described by systems of equations that are similar to those known from high school. Advantages that accrue by an algebraic approach are conciseness, clarity and simplicity; in particular with respect to (computer-aided) calculation rules. The developed algebraic characterisation of hybrid systems allows, for example, the verification of safety aspects by simple algebraic transformations. The approach further enables the use of off-the-shelf *automated theorem provers*. These provers can be employed to verify fundamental properties of hybrid systems.

Over the last decade dozens of logic-based approaches have been applied to hybrid systems. These approaches range from classical propositional logics and modal or temporal logics that have been transferred over to hybrid systems to special logics that have been developed for them. Most of them are well-understood, but due to their different notions, syntax and semantics a uniform treatment is not available. Therefore we explore a compact and uniform treatment for all these logics. The unification is based on the same algebraic techniques as for the characterisation of hybrid systems. This allows cross-reasoning through all unified logics. In this book we pay special attention to the relationship between the logics involved and hybrid systems.

The book presents fundamental methods for the analysis of hybrid systems and results in a coherent family of *algebraic calculi for hybrid systems*. The suitability and the relevance of the theory is proved by first case studies.

Acknowledgement

Thanks to all people who supported me during this work. First of all, I am very grateful to my supervisor Prof. Dr. Bernhard Möller, Universität Augsburg (Germany), for his support, encouragement, valuable comments and discussions. It was and is always a pleasure to work and discuss with him and his group. I also want to thank Prof. Dr. Gunther Schmidt and Prof. Dr. Walter Vogler for reviewing this thesis.

I am also very grateful to Dr. Georg Struth, University of Sheffield (United Kingdom), for fruitful discussions and cooperation during this work. My visit in Sheffield during the academic year 2006–2007 was an exciting and successful time (personally and academically). During my work at this thesis I also visited the University of Queensland (Australia); thanks to Dr. Graeme Smith for interesting research-related discussions.

I am mostly grateful to my colleagues Han-Hing Dang, Prof. Dr. Jules Desharnais, Roland Glück, Prof. Dr. Ridha Khedri, Florian Lautenbacher, Prof. Dr. Martin Müller, Dr. Mark Schäfer and Dr. Kim Solin for countless and fruitful discussions, their valuable remarks on this thesis and joint work.

Moreover, I would like to thank all the members of the Institute of Informatics, Universität Augsburg, and the Department of Computer Science, University of Sheffield, who made the work and the stays enjoyable ones.

For financial support I gratefully acknowledge the German research foundation DFG, the University of Sheffield and the Universität Bayern e.V. In particular, thanks to Prof. Dr. Phil Green, Prof. Dr. Werner Kießling, Prof. Dr. Bernhard Möller, Dr. Georg Struth and Prof. Dr. Gunther Schmidt.

Special thanks are due to my family for their kind support. In particular, I like to thank my father; without his financial support during my studies of mathematics I would have never had the possibility to do these studies and this thesis would never have been written. I am also thankful to my brother. He kindly agreed to produce two complex figures for this thesis and was always willing to listen to my problems that appeared throughout the last years. Last but not least many thanks to my friends for their support and patience during this work.

Chapter 1

Introduction

"Begin at the beginning," the King said gravely,
"and go on till you come to the end: then stop."
L. Carroll (Alice's Adventures in Wonderland)

Hybrid systems arise in every embedded control system when digital controllers or computers are coupled with their environment. This book presents an algebra of hybrid systems. Moreover it develops algebraic calculi for such systems. The algebra itself is based on trajectories and Kleene algebras. The former are standard constructs from mathematics and physics; the latter are fundamental algebraic structures. This book conveys the advantages of algebra to the area of hybrid systems. For example, abstract algebraic reasoning becomes feasible. Even more, one can now apply off-the-shelf first-order automatic theorem provers for verifying safety and liveness properties. We investigate theoretical concepts as well as practical aspects. We show how to describe, analyse and verify hybrid systems. In particular, we discuss properties of the underlying algebra and discuss how concepts from hybrid system analysis can be characterised algebraically, including safety, liveness and Zeno effects. All the presented theory is clarified by examples and finally applied to some case studies.

1.1 Motivation

Nowadays, many systems have to interact with their environment. Examples vary from technical systems, as used in traffic controls and automated manufacturing, to "exotic" applications in fields like chemistry and biology. Hybrid systems are heterogeneous systems characterised by the interaction of discrete and continuous dynamics

and hence a particular kind of reactive systems. Compared to application software, like Microsoft Office® or OpenOffice.org®, hybrid systems have to react immediately if necessary:

An air-bag that inflates to late or never is not acceptable.

Therefore requirements for correctness, liveness and safety are most important issues when building and analysing hybrid systems.

To point out the importance of hybrid systems in life, the IEEE Control System Society has formed a technical committee on hybrid systems (HYSCOM). On the corresponding website, the members state that "the main research efforts in the study of hybrid systems can be categorized in the following sub-areas:

- Synthesis of control schemes, that govern both discrete and continuous actuators and are designed on the basis of a hybrid model of the system, and on-line monitoring schemes that are able to estimate unmeasured quantities and the occurrence of faults (a fault can be interpreted as a switch of dynamic mode).
- Formal verification of safety properties, which aims at certifying that for a given set of initial conditions and a class of exogenous disturbances the hybrid system never enters unsafe areas, and provides counterexamples that can be for instance used by the designer as test-cases in a detailed simulator of the system.
- Stability, robustness, and performance analysis. There is a rich theory and powerful tools for linear systems, nonlinear continuous systems, and for discrete systems, but still a substantial need for results for hybrid systems.
- System theoretical properties, such as existence, uniqueness, persistence of trajectories, observability, reachability, have to be deeply investigated." [HYS]

Often, hybrid systems are too complex for verification tasks using computer-aided tools, like model checking — even with today's powerful computers. During the last decade, various approaches based on formal methods have been developed. Some of the most successful strategies for formal verification use algebraic techniques which describe systems via equation-based formulas. The present book develops such an algebraic approach for hybrid systems and hence is mainly addressed to the second subarea of hybrid systems' studies (see above). Main advantages of algebraic approaches are clarity, conciseness and simplicity, in particular with respect to computer-aided verification.

Over the last decades simple algebraic structures like semirings and Kleene algebras have proved to be fundamental first-order structures in computer science with

widespread applications ranging from program analysis and semantics to combinatorial optimisation and concurrency control. They offer operators for modelling actions, programs or state transitions under non-deterministic choice, sequential composition and finite iteration. There are also extensions for infinite iteration and therefore they are able to model infinite behaviours. Hence such structures seem to be predestinated for hybrid systems.

Based on such a structure we will develop an algebraic view of hybrid systems. In particular, we introduce an algebra for hybrid systems and show how to formalise and specify safety and liveness properties for hybrid systems at an abstract level. This abstraction yields the possibility to use off-the-shelf automated theorem provers for verification. The concrete algebraic model for hybrid systems uses sets of trajectories as elements. Each trajectory corresponds to one possible behaviour of a hybrid system. It is straightforward to give faithful mappings from other formalisms like hybrid automata into our setting. Furthermore, unlike most other approaches, the algebra provides a simple and concise way of modelling Zeno effects. We also present algebraic versions of temporal logics like computation tree logic (CTL^*) and neighbourhood logic. Such an algebraisation allows us to apply logics for reasoning about hybrid systems at an abstract level. Throughout the book we illustrate the theory with several examples. Altogether we derive algebraic calculi for hybrid systems.

1.2 Hybrid Systems

Hybrid systems are heterogeneous systems characterised by the interaction of discrete and continuous dynamics and hence a particular kind of reactive systems. They arise for example in every embedded control system when digital controllers or computers are coupled with continuous systems, like plants. Typically, computers are modelled by finite-state machines and plants by partial or ordinary differential equations. In general, the behaviour of the controller depends on the state and the behaviour of the controlled system and cannot be considered in isolation. More complicated hybrid systems emerge from the composition of many smaller systems.

Hybrid systems appear in almost all disciplines of natural science, all areas of computer science and many technical systems. Traffic controls and traffic management systems [DHO04, FM06] are among the standard applications. In these cases, the traffic (cars, trains, etc.) is described by differential equations and forms the continuous behaviour. Traffic lights and traffic signs that can be modified by a controller are representatives for the discrete part of the system, since they can immediately change their state. Further applications are automated manufacturing (e.g. [Cor88]), computer disk drives [Bro93], constraint robotic systems [BGM93], flight control and flight managing systems [SMT+95] and biological applications [LT04, GT04].

In 1967, Zuse pointed at the importance of hybrid systems within computer science [Zus67]. In his visionary paper he claimed that finite-state machines are the appropriate tool for modelling hybrid systems — even if the treatment of automata with non-discrete states is difficult. Furthermore, he describes the possibilities of cellular automata [vN66], an extension of finite-state machines. A cellular automaton consists of a uniform grid of cells, each being in one of a finite number of states (finite-state machine). Every cell has the same rule for updating, based on the values in his neighbourhood. Every time the rules are applied to the whole grid a new generation is created. He also mentioned that other extensions of finite-state machines might be successful, too. Zuse closes the paper with the statement that his ideas do not offer real solutions at the time of writing, but will give possible and new perspectives and insights.

Finally, about 25 years after that, hybrid systems have been established in computer science. In the 1990ies, Henzinger, Alur and others introduced hybrid automata [ACH+95, Hen96, LSV03]. These automata have, next to nodes and edges, differential equations and variables. These additional features reflect the behaviour of the environment in each node. In fact, hybrid automata can be seen as a generalisation of timed automata [AD94]. Hence hybrid automata combine discrete transition graphs with continuous dynamical systems. They can be seen as infinite-state transition systems, which gives insights into the structure of hybrid state spaces. Moreover, since they are based on finite-state machines, hybrid automata can be understood as a realisation of Zuse's vision.

Over the next years this theory was investigated in detail and extended in several directions [Hen00]. Doing this, safety and invariance properties have gained one of the most attention in studies concerning hybrid systems. It was shown that the reachability as well as safety and liveness analysis of hybrid automata are undecidable [Mil00]. Even for subclasses like linear hybrid automata, where the continuous part must be described by linear equations, these tasks are only semidecidable [SPS00]. In [ACH+95], reachability and verification problems for linear hybrid systems are discussed. Necessary and sufficient conditions for existence and uniqueness of solutions are derived and a class of hybrid automata whose solutions depend continuously on the initial state are characterised in [LJS+03]. In contrast to these papers our approach will not restrict the class of hybrid systems at all.

Since hybrid automata are based on finite-state machines, standard verification techniques like model checking [EC81, CFP00] seem suitable. However, due to the continuous state spaces of hybrid automata, model checking approaches for hybrid system cannot use finite-state abstractions. Therefore various alternatives have been developed, e.g. [HNSY94, CK03, Frä99]; implementations are available. An example for a hybrid system model checker is HyTech [HHWT97], a symbolic model checker for lin-

ear hybrid automata. Another special-purpose model checker is CHARON [AGH+00, ASIM02]. A problem of all these approaches is, like in standard model checking, the state space explosion. Hence verification tools fail already for small systems.

Besides the theory of hybrid automata, logics for hybrid systems have been discussed in detail. In [DN00] Davoren and Nerode give an excellent overview on logic-based formal methods within hybrid systems. The use of linear temporal logic (LTL) and the branching computational logics CTL and CTL* is summarised in [AHLP00, DN00]. Moreover, Davoren and Nerode present a semantics of modal μ-calculus in hybrid system analysis and discuss topological aspects. Another successful verification approach uses Harel's dynamic logic (DL) [Har79, HKT00] which can handle infinite-state systems and operational system models. Particularly with regard to hybrid systems, an extension of DL was developed within the project "Automatic Verification and Analysis of Complex Systems" (AVACS) [Nie], namely the differential dynamic logic (d$\mathcal{L}$) [Pla08]. It captures the logical part of hybrid system dynamics. Neighbourhood logic (NL) [ZH98] is the last logic to be mentioned. Proposed by Zhou and Hansen, it is a complete first-order logic to model hybrid systems. It is based on two novel neighbourhood modalities and it is shown that the logic is complete, adequate and suitable for modelling hybrid systems [Puj97]. All these logical approaches allow deductive verification, but provide only operators for finite iteration. This is a great deficiency. Since most hybrid systems are based on a never ending loop, operators for infinite iteration are necessary.

Deductive verification approaches are used to verify systems by proofs that, as a consequence, avoid state space explosions. In such approaches the above logics have been used to prove validity of formulas using logical calculi (e.g. [DN00, ZRH93]). Tools for deductive verification tools are also available. For example Manna and Sipma [MS98], and Abrahám-Mumm [AM01] verified invariants of hybrid systems in STeP [MS95] and PVS [ORS92], respectively. Since both approaches use higher-order theorem provers, verification tasks usually need a huge amount of user interaction. To eliminate this deficiency, Platzer and Quesel developed the special purpose, first-order theorem prover KeYmaera [PQ08]. Based on the theorem prover KeY and the dynamic logic d$\mathcal{L}$, it is a hybrid verification tool that combines deductive, real algebraic, and computer algebraic prover technologies. It is an automated and interactive theorem prover for a natural specification and verification logic for hybrid systems.

The study of hybrid systems in computer science is still largely focused on hybrid automata and logical-based formal methods. There are only few other approaches. An algebraic framework dealing with hybrid systems is the process algebra of Bergstra and Middleburg [BM05]. It is obtained by extending a combination of two extensions of the algebra of communicating processes (ACP) [BW90], namely the process algebra with continuous relative timing from [BM02] and the process algebra with proposi-

tional signals from [BB97]. It has, in addition to equational axioms, some rules to derive further equations with the help of real analysis. However, it does not contain transformation rules for larger systems; moreover, it does not define operators for the analysis of the finite and infinite parts of behaviours.

In [He94], He presents a variant of timed CSP [SDJ+92] that allows limited dealing with continuous behaviour. In [RS03] the π-calculus [Mil99] is modified by Rounds and Song such that it can deal with continuous behaviour. Last but not least Rönkkö shows a stepwise development of hybrid systems using action systems [Rön01]. But these approaches have not yet been put into algebraic form.

As we will show in the book, algebra not only provides a concise and compact notion for hybrid systems, it also unifies some of the above approaches. For example, we will present algebraic embeddings of $\mathsf{d}\mathcal{L}$, CTL^* and NL. Moreover, the algebra for hybrid systems provides operators for finite and infinite iterations and allows the analysis of finite and infinite behaviours. Hence it solves some of the above deficiencies.

1.3 Kleene and Omega Algebras

Idempotent semirings and their extensions have proved to be some of the most fundamental first-order structures in computer science. They provide operations for modelling actions, programs or state transitions under non-deterministic choice and sequential composition. In most of the applications for idempotent semirings iteration is needed and plays a crucial rôle.

For many applications an operator for finite iteration is sufficient. Often, such an operator is based on the Kleene star and on Kleene algebra. Its theory starts probably with a technical report by Kleene in 1951 [Kle51] which was later published in a shortened and revised version in [SM56]. Influenced by biological questions Kleene develops the theory of regular events. He defines tables for nerve nets and neuron activities and gives operations for composition, sum and iteration. He specifies the regular set of tables as the least class (fixed point) that includes the unit and is closed under composition. Last, he puts the challenge to find an axiomatisation of regular events.

In 1964, Redko showed that there is no finite, equational axiomatisation [Red64]. Nevertheless, various axiomatisation have been developed and proposed. Two sound and complete axiomatisations for regular events over a finite alphabet were given by Salomaa [Sal66]. Later on, Bloom and Ésik as well as Krob presented axiomatisations with infinite numbers of equations [BÉ93, Kro91]. Conway was one of the first researchers who studied the algebra of regular events extensively [Con71]. He presented various axiomatisations of Kleene algebra, compared their expressiveness and estab-

lished the connection between the algebra and finite-state machines. In 1990, Kozen improved Conway's axiomatisations and gave a concise axiomatisation. He defines a Kleene algebra as an idempotent semiring equipped with a special operator for finite iteration which is characterised by least fixed points [Koz94] and axiomatised by Horn clauses. He also showed that the axiomatisation is complete for the equational theory of regular expressions. Consequently, all regular identities hold in Kleene algebras. Initially conceived as algebras of regular events, they now find widespread applications. Kuich and Salomaa summarised applications within the theory of finite-state machines in [KS86]. Applications within this area vary from graph theory [SS93] to simple game theory [BM04]. Further applications of Kleene algebra range from program analysis and semantics (e.g. [DMS06]) to combinatorial optimisation, concurrency control [Coh94, HMSW09] and pointer structures [Ehm04, Möl97]. Standard models of Kleene algebras are, among regular languages, all standard models of relation algebra (e.g., [Tar41, SS93, Mad06]). In such algebras the Kleene star coincides with the reflexive and transitive closure.

In many cases finite iteration modelled by the Kleene star is sufficient, but often a characterisation for infinite iteration is necessary, e.g., for modelling non-termination. As we will see, hybrid system analysis is another classical example. To model infinite behaviour Kleene algebras are extended in different ways. In [Coh00], Cohen introduce an omega operator which is intended to model infinite iteration on the basis of Kleene algebra. In 1980, Park also presented an algebraic operator [Par80]. His algebraic approach uses the same axiomatisation as Cohen's omega algebra and can therefore be seen as one of the first attempts to model infinite behaviour algebraically in computer science. The omega operator bears strong similarities to Büchi automata [BD06]. Thus it seems to be adequate to model infinite behaviour. It was successfully applied in reasoning about concurrent programs, progress and termination [DMS04b, Str06]. Although an analysis of partial correctness of programs is possible, an analysis with respect to total correctness causes problems. Next to that and contrary to intuition, the omega operator shows anomalies in some very natural models [HS08b], including regular languages.

At about the same time when Cohen developed his omega algebra, von Wright used a slightly different approach to combine the advantages of Kleene algebra with the refinement calculus [vW02, vW04]. He successfully applied the resulting demonic refinement algebra to several fields, including total correctness reasoning, program transformations and data refinement. Essentially, a demonic refinement algebra is a Kleene algebra, where the axiom for right annihilation ($a \cdot 0 = 0$) is dropped and which is equipped with an additional operator, called strong iteration. Such an operation appeared already in earlier work of Back and von Wright [BvW98, BvW99], where they study concrete models, namely conjunctive predicate transformers. In contrast to the omega operator, strong iteration models finite or infinite iteration, i.e., it is

an iteration which either terminates or goes on forever. In [Möl04, Möl07], Möller investigates a relaxation of Kleene algebra, where he gave up two axioms: the right annihilation and right distributivity. His approach subsumes Dijkstra's computation calculus, Cohen's omega algebra and von Wright's demonic refinement algebra. In 2006, Höfner, Möller and Solin showed how Kleene algebra and refinement algebra are explicitly related [HMS06].

Another extension of Kleene algebra occurred in reasoning about computer programs. There, concepts like (pre-/post-) conditions play a significant rôle. Among others some of the most important contributions for such conditions are certainly Hoare logic [Hoa69] and (based on that work) Dijkstra's computation calculus [Dij75]. Pre- and postconditions can be seen as test programs (e.g., [Pra92]). Such a program has to preserve the current state if and only if the condition is satisfied; otherwise it has to stop the complete program. In [Koz97], Kozen gives an approach to capture tests algebraically on the basis of Kleene algebra. A Kleene algebra with tests has an embedded Boolean subalgebra; hence such an algebra is two-sorted. Kozen also showed how to express Hoare logic within this setting [Koz00]. Desharnais, Möller and Struth extend Kleene algebra with tests by a domain operator which links algebraic, relational and modal approaches; it provides equal opportunities for propositions and actions [DMS06]. The three simple equational domain axioms allow the definition of modal operators semantically via abstract image and preimage operations [DMS04a, MS06]. In many cases, expressions that mention modalities can be reduced to pure Kleene algebra with tests. This preserves the algorithmic complexity and provides a very symmetric approach to reasoning about actions and propositions or transitions and states. The resulting formalism bears strong similarities with propositional dynamic logic [HKT00]. Using such a structure one can now reconstruct Noethericity and well-foundedness in an algebraic way. However, the resulting formalism is still two-sorted.

A two-sorted approach seems to be quite unnatural. Recently, Desharnais and Struth eliminate this deficiency and showed that the domain operator can be generalised and therefore the underlying Boolean algebra does not need an extra sort [DS08].

So far, Kleene algebras have found widespread applications, providing a uniform semantics for various program analysis tasks. They support cross-theory reasoning with modal, relational and language-based approaches. They come with a simple first-order equational calculus that, by experience, yields particularly short and abstract proofs. Kleene algebras have already been integrated into higher-order theorem provers [Str02, Kah04, AHK06], and their applicability as a formal method has successfully been demonstrated in that setting. Recently, Höfner and Struth explored their potential for first-order automated deduction. In [HS07] Kleene algebras are axiomatically implemented in Prover9/Mace4 [McC] and proof experiments about

Hoare, dynamic, temporal logics, concurrency control and termination analysis are performed. They confirm that often a fully automated analysis of important program properties is feasible. Afterwards automated reasoning was successfully applied for reasoning in variants of Kleene algebra, including demonic refinement algebra [Str07, HS08a].

Based on Kleene and omega algebras we will develop an algebra for hybrid systems. Since all regular identities hold in Kleene algebras, this algebra is the algebraic counterpart of regular languages and finite-state machines. It seems to be a promising candidate for hybrid automata (based on finite-state machines) and therefore for hybrid systems. In this book we will show that Kleene algebra is at least one possible basis for building an algebra for hybrid systems; but it has to be modified to cover the specific characteristics of such systems, namely infinite iteration and infinite behaviours.

1.4 Contribution and Organisation

The contribution of the book consists mainly of two parts. Based on semirings, we first derive an algebra of hybrid systems. To achieve this, we relax the algebraic structure of semirings and introduce lazy semirings. This is the appropriate algebraic structure for hybrid systems. Based on this algebra we investigate basic properties for hybrid systems, including operators for finite and infinite iteration, and aspects of safety and liveness. In the second part we derive closed semantic expressions for branching time logic and neighbourhood logic. Since the structures underlying these logics and the algebra of hybrid systems coincide, we are able to do cross-reasoning and to apply logics to hybrid systems algebraically.

The book is organised as follows:

Chapter 2 introduces basic concepts for hybrid systems. In particular, we recapitulate the definition of hybrid automata and show how such automata can be composed. Moreover, we give a couple of examples to illustrate the variety of such systems and to clarify the notions and definitions introduced.

Chapter 3 first recapitulates the basic notion of semirings, Kleene algebras and omega algebras. After that, we show why these structure are too restrictive and cannot model hybrid systems properly. After defining the basic concepts of lazy semirings (lazy Kleene and omega algebra), we investigate basic properties of these structures. In doing so we put the main focus on properties that are necessary for hybrid system analysis and on proof automation. In particular we show that the algebraic structures used are suitable for fully automated theorem provers (ATP systems). A proof script not only acts as a "certificate" and increase the confidence in the result, but an auto-

mated theorem prover tool can fully automate many required proofs and therefore safe a lot of time and effort.

The first main contribution is given in **Chapter 4** where an algebra of hybrid systems is defined. In particular, we show that sets of possible behaviours (trajectories) satisfy the axioms of lazy semirings, lazy Kleene and lazy omega algebras. We investigate how properties of hybrid systems, such as safety and liveness can be formalised algebraically. For that, we also define useful operators. Moreover we discuss how the composition of hybrid automata can be simulated in the algebraic setting. This algebra paves the way to use ATP systems for hybrid system analysis.

Chapter 5 is concerned with logics; the second main contribution of the book. We derive closed semantics for the branching time logic CTL^* and for neighbourhood logic. The underlying algebraic structures are again lazy semirings, lazy Kleene algebras and lazy omega algebras. Since both, the algebra of hybrid systems and the algebraisation of these logics, are based on the same framework, cross-reasoning is possible and there is nearly nothing to do to apply the logics to hybrid systems.

Throughout the book we illustrate the introduced theory by small examples. In **Chapter 6** we supplement these with some larger case studies: We revisit an example from Chapter 2 that models a controller for a railway crossing gate. Furthermore we automatically check a given specification for a security service. Last but not least, we verify a liveness condition for an assembly line scheduler.

Finally, **Chapter 7** gives a conclusion of the book and discusses some future research.

In the **Appendix**, we summarise the results of proof automation and provide templates in Prover9- and TPTP-style for the structures involved. Hence the interested reader can easily use arbitrary ATP systems in the area of lazy semirings. Moreover we present some deferred properties and proofs and some less essential details for the railway crossing gate controller.

Chapter 2

Hybrid Systems — A Short Introduction

Εν χρόνω γὰρ πᾶσα κίνησις καὶ τέλονς τινός.
Aristotle (Nicomachean Ethics X)

Hybrid systems are systems composed of continuous and discrete components. Continuous models are typically associated with physical principles, such as thermo- or fluid dynamics, or classical mechanics. In contrast to that, discrete components are used to model logic devices, such as switches, digital circuits or software code. In computer science hybrid systems are usually represented by hybrid automata or logical expressions.

This chapter provides the basics for characterising hybrid systems. More precisely, we recapitulate the definition of hybrid automata and discuss concepts such as trajectories and runs. In Chapter 4, sets of trajectories will be used to develop an algebra of hybrid systems. Since more complicated and more complex hybrid systems often emerge from the composition of smaller systems, we also show how to compose hybrid automata.

In this chapter we do not yet discuss logical expressions for hybrid systems; this will be done in Chapter 5.

2.1 An Introductory Example

The most elementary and classical hybrid system usually consists of one controlling subsystem, the controller for short, and one controlled subsystem. We start with such

an elementary system as an introductory example. It models a thermostat which has control over a heating of a room (adapted from [Hen96]). It can only affect the temperature of the environment. If the room temperature falls below a given value, the heating has to be switched on; if the temperature keeps rising, the heating has to be switched off at some point. We model this system using an hybrid automaton. Such automata combine finite-state machines with differential equations. An exact definition will be given in the next section.

A possible hybrid automaton for the temperature control is shown in Figure 2.1. It consists of only two nodes (*control modes*), On and Off. The edges between the nodes reflect changes in the discrete behaviour. Here, the upper one stands for switching the heating on, the lower for turning the heating off. The initial state is Off.

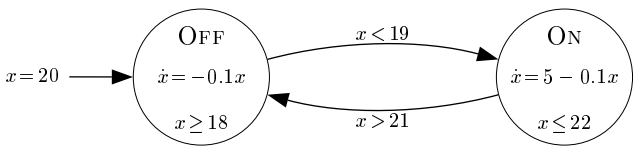


Figure 2.1: Thermostat automaton

The variable x represents the temperature. Initially, the temperature is 20° Celsius. The temperature decreases according to laws of thermodynamics described by a *flow condition* formalised by the differential equation $\dot{x} = -0.1\,x$. If the *jump condition* $x < 19$ is reached, the heater may start. The *invariant condition* $x \geq 18$ ensures that the heater will start at the latest when the temperature is equal to 18 degrees. In control mode On, the temperature rises according to the flow condition $\dot{x} = 5 - 0.1\,x$. If the temperature is between 21° and 22°, i.e, the temperature satisfies the corresponding jump and invariant conditions, then the heater is switched off and the procedure starts again (with another initial value).

As already mentioned, the continuous part of the system behaviour is usually given by differential equations. However, looking at the example, we see that the differential equations can be solved immediately. Separation of variables (e.g., [Arf85]) yields the solutions of the flow condition of the modes Off and On:

$$f^{\text{Off}}(t) \;=\; x_0 \cdot \exp\left(\frac{t_0}{10}\right) \cdot \exp\left(\frac{-t}{10}\right) \quad \text{and}$$

$$f^{\text{On}}(t) \;=\; (x_0 - 50) \cdot \exp\left(\frac{t_0}{10}\right) \cdot \exp\left(\frac{-t}{10}\right) + 50\,,$$

where x_0 is the initial value of the temperature at time t_0. For the above automaton, the initial values are for example $t_0 = 0$ and $x_0 = 20$, i.e., the function becomes

$20 \cdot \exp\left(\frac{-t}{10}\right)$. With these equations one can easily determine possible temperature curves that are accepted by the hybrid automaton. One possible curve, which is also called a *run* or a *trajectory*, is sketched in Figure 2.2.

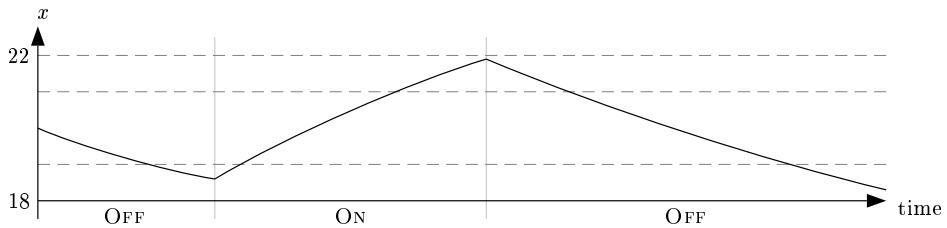


Figure 2.2: An accepted run of the temperature controller

Next, we illustrate the concepts of *safety* and *liveness* . Informally, a system is safe if "something bad will never happen" [Lam77]. In our example a temperature above 50° can be regarded as bad. Since the temperature will stay between 18° and 22° the system is safe with respect to that property.

Informally, a system is alive, if "something good will eventually happen" [Lam77]. In our example a liveness criterion may be that the temperature will finally reach a temperature between 18° and 22° independently of the initial temperature. This criterion holds at least if the initial temperature satisfies the invariant condition of Off. More often, liveness properties are used to verify that a given system will not get stuck. To guarantee that the temperature control is alive, one can postulate that the system will be switched on again after it was switched off and vice versa. This ensures that the system is switched on/off infinitely often and therefore the system will not stuck.

Finally, we show that the continuous behaviour of a system is not limited to smooth behaviours. For this, we extend the above temperature controller by an additional switch sw that allows one to activate or deactivate the whole temperature control. The switch can only be in two modes: the value $sw = 1$ represents the situations where the whole temperature control is activated, and $sw = 0$ means that the heating system is deactivated. When the switch is toggled the continuous behaviour jumps from 0 to 1 or vice versa. This example contains proper jumps (in the behaviour of the switch) as well as "non-proper" jumps (in the change of temperature). The whole system is shown in Figure 2.3.

The system can always be deactivated by setting the switch to 0 (independent of the current temperature). When reactivating the system there is a choice between the modes Off and On. It is a genuine non-deterministic choice if the temperature

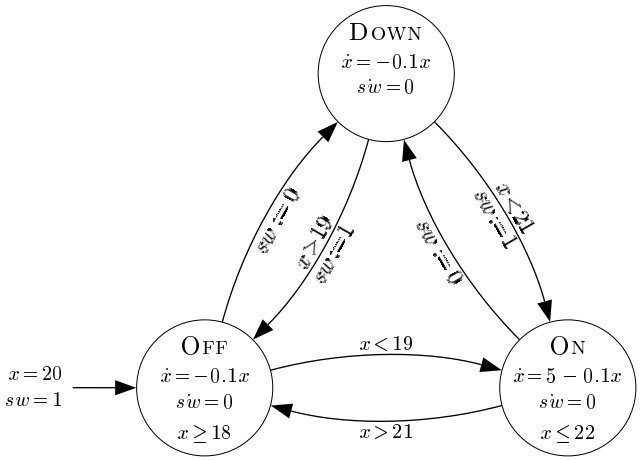


Figure 2.3: Extended thermostat automaton

is between 19 and 21 degrees. To improve readability, conditions of constancy like $\dot{sw} = 0$ are often omitted.

2.2 Basic Concepts

Following the above example we now define the fundamental concepts for hybrid systems. Furthermore, we introduce some notational conventions.

The above example shows that hybrid automata have, next to nodes (corresponding to states) and transition edges, variables and differential equations (cf. Figure 2.1). Since the differential equations are assigned to the nodes, they reflect the behaviour of the environment in each control mode. In fact, hybrid automata can be seen as a generalisation of timed automata [AD94].

Definition 2.2.1 (hybrid automaton [ACHH93, DN00, Hen96])
A *hybrid automaton* H consists of the following components:

VARIABLES. A finite set $X = \{x_1, \ldots, x_n\}$ of real-valued variables. The number n is called the *dimension* of H. We write $\dot{X}$ for the set $\{\dot{x}_1, \ldots, \dot{x}_n\}$ of dotted variables, which represent the timewise first derivatives of the x_i during continuous change. We write X' for the set $\{x'_1, \ldots, x'_n\}$ of primed variables, which represent the values of the x_i immediately after a discrete change.

CONTROL GRAPH. A finite directed multigraph (M, E). The vertices in M are called *(control) modes*. The edges in E are called *(control) switches*.

JUMP CONDITIONS. An edge labelling function *jump*. It assigns to each control switch $e \in E$ a predicate $jump(e)$ with free variables from $X \cup X'$.

INVARIANT AND FLOW CONDITIONS. A vertex labelling function *inv* and a vertex labelling function *flow*. They assign to each control mode $v \in M$ an invariant $inv(v)$, a predicate with free variables from X, and a flow condition $flow(v)$, a predicate with free variables from $X \cup \dot{X}$.

INITIAL CONDITION. The vertex labelling function *init* assigns to at least one control mode $v \in M$ an initial condition $init(v)$, a predicate with free variables from X.

The control graph describes, together with the jump conditions, the discrete behaviour, whereas the other components are used to model the continuous behaviour. A particular feature of such automata is that they do not have accepting states like finite-state machines, because there is no end of the continuous behaviour and therefore the controller should operate "forever". If a control mode does not contain flow conditions for the variable x_i we assume the variable to be constant, i.e., the mode implicitly contains the differential equation $\dot{x}_i = 0$. An edge (switch) leading from mode v to mode w is also called a *transition* $t_{v,w}$. The hybrid automaton *can perform* such a transition if the end values X of mode v and the starting values X' of mode w satisfy the predicate $jump(t_{v,w})$. The hybrid automaton *must perform* a transition before violating the invariant condition. The conditions for a mode have to be satisfied even if no time passes when using it, i.e., a zero-duration trajectory occurs (see below). This can happen if the mode acts as an intermediate changing point to perform a series of transactions from one mode to another. In the case that the automaton is in a situation where it must perform a transition, but cannot perform any, the whole system crashes, i.e., the automaton cannot accept any run which at some point reaches such a situation. Therefore the invariant and jump conditions should be chosen carefully. A transition is called a *proper jump* if it changes at least one value $x \in X$ to a new value $x' \in X'$ with $x \neq x'$.

In the original definition Henzinger introduces another edge labelling function which assigns an event to each control switch. This function does not reflect a part of a hybrid system. It is a man-made function that allows to model parallel composition. In Section 2.4, we will use a relational approach to characterise parallel composition. From this, the edge labelling function will arise in a natural way.

Example 2.2.2 Formally, the hybrid automaton for the temperature control of Figure 2.1 is defined by the set of variables $X = \{x\}$, the control modes $M = \{\text{OFF}, \text{ON}\}$,

the control switches $E = \{(\text{OFF}, \text{ON}), (\text{ON}, \text{OFF})\}$. The invariant function $inv(v)$ assigns $x \geq 18$ to mode OFF and $x \leq 22$ to ON. The flow condition $flow(v)$ is $\dot{x} = -0.1x$ inside mode OFF and $\dot{x} = 5 - 0.1x$ inside ON. An initial condition exists only for the mode OFF and sets the value of x to 20. Finally, the jump conditions are defined by $x < 19$ for the edge (OFF, ON) and $x > 21$ for (ON, OFF). □

With each hybrid automaton one can associate traces, runs and trajectories. We will later use these concepts to define our algebra of hybrid systems.

Definition 2.2.3 (transition trace, (mode) trace [vM99])
A *transition trace* of a hybrid automaton is a (finite or infinite) sequence of transitions $t_{v_k, v_{k+1}}$ which the hybrid automaton can perform as time passes (forever). The *(mode) trace* of a hybrid system corresponding to a transition trace is the sequence $v_0, v_1, \ldots$ of modes through which the transition trace passes.

Since hybrid systems have no accepting states, they have to work forever. This implies that transition and mode traces can only be finite if and only if the system is able to stay in one control mode for an infinite amount of time without violating the invariant condition.

Definition 2.2.4 (run, trajectory)
A *trajectory* or *run* is a function from an interval of $\mathbb{R}_{\geq 0}$ to n-tuples of values for all n variables.[1] It corresponds to a trace (and a transition trace) if the underlying interval is $[0, \infty[$.

According to this definition, trajectories reflect the variation of the values of the variables over time. A trajectory is called *infinite* if it is defined on the interval $[x, \infty[$, otherwise it is called *finite*. Informally, a trajectory is infinite if it lasts forever.

Example 2.2.5 One possible trajectory for the thermostat has been already given in Figure 2.2. Each part (restriction to an arbitrary interval) of this trajectory is also a trajectory. For example, the trajectory between the two vertical gray lines of Figure 2.2 describes a possible internal behaviour of the control mode ON.

The only possible (infinite) mode trace is the infinite alternating sequence

$$\text{OFF ON OFF } \ldots \text{ .}$$

There is no finite sequence since the temperature falls steadily in mode OFF and therefore the heating is forced to switch into mode ON. (Otherwise the invariant

[1] In Chapter 4, we define trajectories over a generalised time domain.

condition $x \geq 18$ would be violated.) Vice versa, in control mode On the temperature would steadily rise and the invariant condition would be violated if the transition $t_{\text{On,Off}}$ were not performed.

A finite trace can be found in the extended heating corresponding to Figure 2.3, since the automaton is not forced to leave mode Down. Examples are "Off On Down" or "Off On Off Down Off On Down". □

2.3 More Examples

Hybrid systems appear in almost all areas from physical and technical systems to biological processes. To give an impression of the variety of such systems we provide some more examples from different disciplines. Furthermore we will revisit most of these examples to underpin the presented theory. We often omit the formal definition of the involved hybrid automaton and give only the graphical representation.

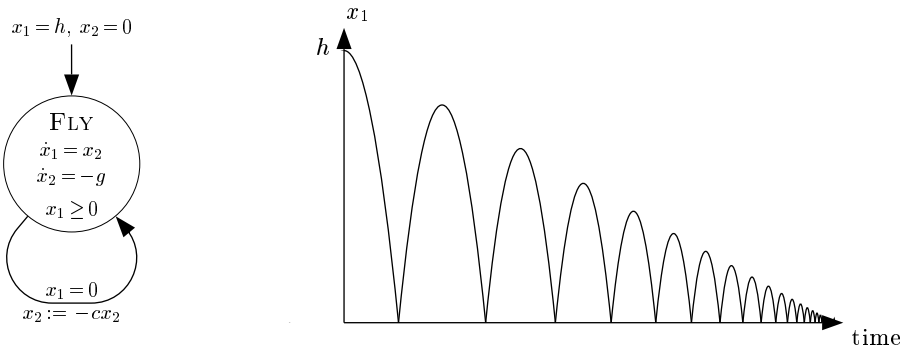


Figure 2.4: Hybrid automaton of a bouncing ball and corresponding run w.r.t. x_1

Bouncing ball. Our next example describes a bouncing ball. It is, next to the temperature control, one of the canonical examples in the literature. A ball which is assumed to be a point-mass falls from an initial altitude and bounces back from the ground, losing part of its energy when touching the ground. Between each bounce, the behaviour of the ball is described by differential equations; hence this is the continuous part of the hybrid system. The discrete part occurs when the ball touches the ground and its velocity changes immediately (modelled by an inelastic collision). The corresponding hybrid automaton and the only existing trajectory (restricted to x_1) is given in Figure 2.4.

The variable x_1 represents the altitude of the ball, x_2 its velocity. The initial altitude is h and the initial velocity is 0. If the ball is above the ground ($x_1 > 0$), its flow is governed by $\dot{x}_1 = x_2$, $\dot{x}_2 = -g$, where g is an arbitrary positive gravity force. These equations state that when the ball is above the ground, it is being drawn to the ground by gravity. Moreover, we assume a damping factor $0 \leq c < 1$ which makes the ball lose energy with every bounce. This example is an especially interesting one, as it contains Zeno behaviour (cf. Section 4.4). Zeno behaviour has a strict mathematical definition, but can be described informally as the system making an infinite number of jumps in a finite amount of time. In this example, the loss of energy makes the subsequent jumps closer and closer together in time (cf. right part of Figure 2.4).

Gear shift control ([Lyg04]). This example models a gear box of a car with four gears. Depending on the selected gear, the car is slower or faster and changes its position (x_1). For simplicity we assume that this simple car can only move in one direction (no direction control).

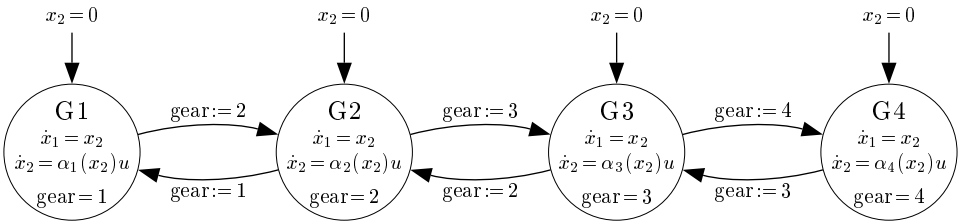


Figure 2.5: Hybrid automaton of a car with four gears

The longitudinal position of the car is denoted by x_1, its velocity by x_2, the throttle position by $u \in [u_{\min}, u_{\max}]$, and the function α_n represents the efficiency of gear n. Each control mode is an initial one. In our example the example trajectories are only accepted by the hybrid automaton if there is no velocity at the beginning. A typical trajectory is given later in Section 4.6 (Figure 4.9).

To round off the picture of possible applications, we give an example from systems biology. Since we focus on engineering examples in this book, we sketch an example from high school. More complicated examples can be found e.g. in [GT04] where the Delta-Notch protein signalling is described or in [LT04] where biological and biochemical networks are analysed. Such networks describe the interaction of genes, proteins and other molecules.

Cell-devision cycle. The cycle is a series of phases of a cell that leads to cell replication. It can be split into two parts: The *interphase*, where the cell grows, nutrients needed for mitosis are accumulated and its DNA is duplicated, and the *mitosis* (M), during which the cell splits itself into two distinct cells. The interphase is divided into three sections: In G_1 the cell grows; in S the synthesis occurs, that is the genome is replicated, and in G_2 the cell continues growing and prepares itself for division.

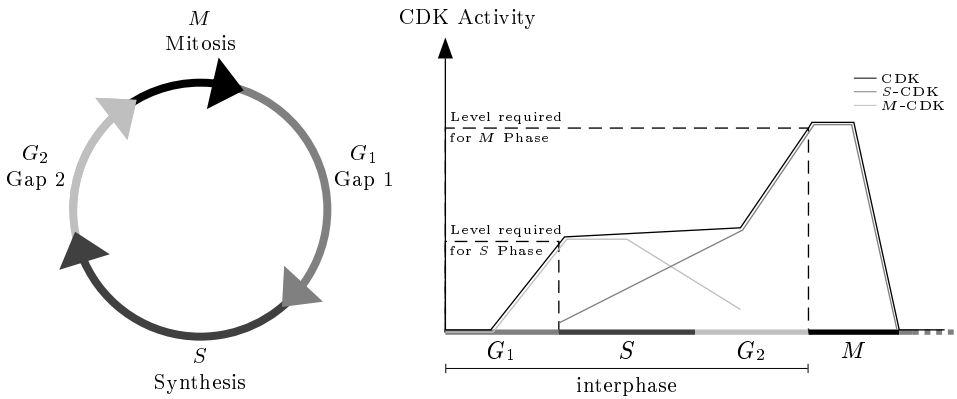


Figure 2.6: Cell-devision cycle and trajectories for CDK

A hybrid automaton can be immediately extracted from the left part of Figure 2.6; each phase symbolises one control mode, the switches between the phases are the discrete switches.

Two classes of regulatory molecules bring the cell from one cell cycle phase to the next: cyclins and cyclin-dependent kinases (CDKs) [Nig95]. The amount of CDK-molecules is constant during the cell cycle, but their activities vary because of the regulatory function of the cyclins. Therefore CDK-molecules can be used to model progress and to find the timepoints when to perform transactions.

In the right part of Figure 2.6 the quantitative model of cell cycle regulation is illustrated [SN96, DLL$^+$06]. There the trajectories for different CDKs are presented. Since we want to give only a rough idea of the hybrid system, we omit the complicated differential equations for the concentration of the diverse molecules. Of course, one may add much more information to the modes which makes the model more accurate but also more complicated.

2.4 Composing Hybrid Systems

So far, we gave only small representants for hybrid systems. More complicated hybrid systems emerge from the composition of several smaller systems. The product of two finite state machines as well as the parallel composition are well known. Similar to these constructions a product and a parallel composition for hybrid automata can be defined [Hen96]. In this section, we will briefly recapitulate these constructs. A discussion in more detail is given in [BS98].

For the forthcoming definitions, we introduce additional notions. Assume an arbitrary control switch e (an edge of the underlying multigraph), then ${}^\bullet e$ denotes its *source* and $e^\bullet$ its *target*.

Definition 2.4.1 (product of hybrid automata)
For two hybrid automata H_1, H_2 the product hybrid automaton $H =_{df} H_1 \times H_2$ is defined as follows.[2]

- The set of variables is the union of all variables of H_1 and H_2, i.e., $X_1 \cup X_2$, where we assume that X_1 and X_2 are disjunct.
- The set of control modes is $M_1 \times M_2$. Without loss of generality we assume that the sets of control modes are disjoint, i.e., $M_1 \cap M_2 = \emptyset$.
- There is a control switch between two control modes if and only if there is at least one edge for each single automaton, i.e.,

  $$(({}^\bullet e_1, {}^\bullet e_2), (e_1^\bullet, e_2^\bullet)) \in E \quad \Leftrightarrow_{df} \quad e_1 \in E_1 \wedge e_2 \in E_2 \ ,$$

  where E denotes the set of all control switches of H.
- The remaining conditions (*jump*, *inv*, *flow* and *init*) are defined via the Cartesian product in a straightforward way. For example, for all $v_1 \in M_1$ and $v_2 \in M_2$, $\mathit{inv}((v_1, v_2)) =_{df} \mathit{inv}(v_1) \wedge \mathit{inv}(v_2)$.

The product construction is not very useful, since it determines all possible combinations of switches and hence is total synchronisation. In particular, it does not consider specific synchronisation at all. Performing a transition in one automaton and remaining in the current state in the other is not possible. As we will see, such behaviour is necessary for hybrid systems. More often parallel composition of hybrid systems is used for the specification of larger systems. The communication between

[2] In the sequel, components of H_1 are denoted by index 1, components of H_2 have index 2.

the components may occur via shared variables and synchronisation labels. We assume synchronisation labels only at control switches.

In the sequel we will define two different versions of parallel composition. Both are based on the product construction. The first forces transitions of one automaton *to wait* until synchronisation with a transition of the other automaton is possible (if at least one synchronisable transition exists). In the second version, transitions must only be *synchronised when possible*. If no synchronisation is possible at the moment any transition with valid jump condition can be performed. The first variant follows roughly the definition of Henzinger ([Hen00]). However, the original definition makes several implicit assumptions. Therefore we use another relational approach to parallel composition from which his definition arises in a formal and concise way.

Both constructions use a composition relation that identifies the switches that should be synchronised.

Definition 2.4.2 (composition relation, synchronisable switch)
A *composition relation* $\|$ couples the control switches of two hybrid automata H_1 and H_2, i.e., $\| \subseteq E_1 \times E_2$.

A control switch e_1 of H_1 is *synchronisable* iff there is a switch e_2 of H_2 with $e_1 \| e_2$.

The first variant of parallel composition is defined as follows. Two hybrid automata H_1 and H_2 interact via coupled control switches. A switch e_1 of H_1 can be performed at any time if there is no synchronisable transition in H_2. In this case the other automaton stays in its current control mode. If there is at least one synchronisable transition in H_2, e_1 can only be performed if, at the same time, a synchronisable transition in H_2 is performed.

Definition 2.4.3 (strict parallel composition)
The parallel composition $H_1 \|_s H_2$ of two hybrid automata H_1 and H_2 with respect to a given composition relation $\|$ is defined as follows.

- The set of variables is again the union of all variables of H_1 and H_2, i.e., $X_1 \cup X_2$, where we assume that X_1 and X_2 are disjoint.
- The set of control modes is $M_1 \times M_2$. Without loss of generality we assume that the sets of control modes are disjoint, i.e., $M_1 \cap M_2 = \emptyset$.
- There is a control switch (an edge) from control mode (v_1, v_2) to (w_1, w_2) if
  - there are synchronisable switches from v_1 to w_1 and from v_2 to w_2. The set of all such edges is defined by

$$F_s =_{df} \{((^\bullet e_1, {}^\bullet e_2), (e_1^\bullet, e_1^\bullet)) \,|\, e_1 \in E_1,\, e_2 \in E_2,\, e_1 \| e_2\} \,.$$

– there is a control switch $(v_1, w_1) \in E_1$ and there is no synchronisable switch in E_2. In this case, the transition t_{v_1,w_1} can be performed while the system remains in v_2. The corresponding set of all such edges is defined by

$$G_s =_{df} \{((^\bullet e_1, v_2), (e_1^\bullet, v_2)) \,|\, e_1 \in E_1,\ v_2 \in M_2,\ \nexists\, e_2 \in E_2 : e_1 \parallel e_2\} \ .$$

– there is a control switch $(v_2, w_2) \in E_2$ and there is no synchronisable switch in E_1. In this case, the transition t_{v_2,w_2} can be performed while the system remains in v_1. The corresponding set of all such edges is defined by

$$H_s =_{df} \{((v_1, ^\bullet e_2), (v_1, e_2^\bullet)) \,|\, e_2 \in E_2,\ v_1 \in M_1,\ \nexists\, e_1 \in E_1 : e_1 \parallel e_2\} \ .$$

The set E of all control switches of $H_1 \parallel_s H_2$ is then given by the union

$$E =_{df} F_s \cup G_s \cup H_s \ .$$

- The jump condition for an edge depends on which of these sets the switch lies in.

$$\begin{aligned} jump(e_1) \wedge jump(e_2) \quad &\text{if} \quad ((^\bullet e_1, ^\bullet e_2), (e_1^\bullet, e_2^\bullet)) \in F_s \ , \\ jump(e_1) \quad &\text{if} \quad ((^\bullet e_1, v_2), (e_1^\bullet, v_2)) \in G_s \ , \\ jump(e_2) \quad &\text{if} \quad ((v_1, ^\bullet e_2), (v_1, e_2^\bullet)) \in H_s \ , \end{aligned}$$

  where $e_1 \in E_1$, $e_2 \in E_2$, $v_1 \in M_1$ and $v_2 \in M_2$.

- The mode conditions (*inv*, *flow*, *init*) are defined via the cartesian product. For example $inv((v_1, v_2)) =_{df} inv(v_1) \wedge inv(v_2)$ for all $v_1 \in M_1$ and $v_2 \in M_2$.

An example for three small hybrid systems that are composed in parallel by the above construction is given in the next section.

In [Hen96, Hen00] the case for shared variables is not discussed. Since we only want to recapitulate the basics of hybrid systems, we skip this topic, too.

The composition relation is closely related to Henzinger's edge-labelling function *event*. In [Hen96], this function is used to enforce synchronisation. To indicate the composition relation in hybrid automata we also use an edge-labelling function. Whenever we do so, we mean the corresponding relation $\parallel$. Formally, the relationship is given by

$$event(e_1) = event(e_2) \quad \Leftrightarrow_{df} \quad (e_1, e_2) \in \parallel \ ,$$

where e_1 is an edge of H_1 and e_2 an edge of H_2. In the presented examples this relationship yields no problems; however, if one wants to define a general relationship one has to consider multiple occurences of the same label in the same automaton.

As mentioned before a performable transition has to wait until synchronisation is possible. In some situations this seems too restrictive. In the next section we will give a concrete example describing a small railway system. The following example is construed in a way to demonstrate the main issue.

Example 2.4.4 We assume a hybrid automaton H_1 that consists of two modes v_1 and v_2 and a single switch e_1 from v_1 to w_1 with $jump(e_1) = j_1$. The initial mode is v_1. We do not care about all other parts of the automaton and leave them unspecified here. Furthermore we assume another similar automaton H_2 with modes v_2 and w_2, edge e_2 and $jump(e_2) = j_2$. The only difference is that the initial mode is w_2.

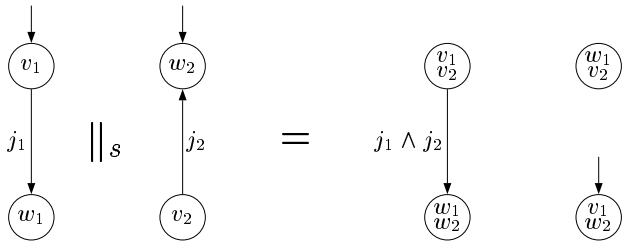


Figure 2.7: Strict parallel composition of two simple hybrid automata

The parallel-composed automaton $H_1 \|_s H_2$ cannot perform a single transitions, if the two existing switches are synchronisable. □

Due to this, we relax the synchronisation constraints. Again we relate transitions of the two automata H_1 and H_2 by a composition relation. Intuitively, if these switches are coupled and H_1 and H_2 are able to perform the respective transitions from the current mode, either both hybrid automata perform the respective transition or none. If the switches are not coupled then either H_1 or H_2 can perform a transition; the other automaton stays in its current control mode.

Definition 2.4.5 (liberal parallel composition)
Except for the control switches and the jump conditions, the liberal parallel composition $H_1 \|_l H_2$ of two hybrid automata H_1 and H_2 with respect to a given composition relation $\|$ is identical to the definition of strict parallel composition (Definition 2.4.3). The set E of all edges of the composed automaton $H_1 \|_l H_2$ is again defined by case distinction.

- There is a control switch (an edge) from control mode (v_1, v_2) to (w_1, w_2) if

– there are synchronisable switches from v_1 to w_1 and from v_2 to w_2. The set of all such edges is defined by

$$F_l =_{df} \{((^\bullet e_1, {}^\bullet e_2), (e_1^\bullet, e_1^\bullet)) \mid e_1 \in E_1,\, e_2 \in E_2,\, e_1 \parallel e_2\} .$$

– there is a control switch $(v_1, w_1) \in E_1$ and there is no synchronisable switch in E_2 that starts in v_2 (with ${}^\bullet e_2 = v_2$). The set of all such edges is defined by

$$G_l =_{df} \{((^\bullet e_1, v_2), (e_1^\bullet, v_2)) \mid\ e_1 \in E_1,\, v_2 \in M_2, \nexists\, e_2 \in E_2 : {}^\bullet e_2 = v_2 \wedge e_1 \parallel e_2\} .$$

– there is a control switch $(v_2, w_2) \in E_2$ and there is no synchronisable switch in E_1 that starts in v_1 (with ${}^\bullet e_1 = v_1$). The set of all such edges is defined by

$$H_l =_{df} \{((v_1, {}^\bullet e_2), (v_1, e_2^\bullet)) \mid\ e_2 \in E_2,\, v_1 \in M_1, \nexists\, e_1 \in E_1 : {}^\bullet e_1 = v_1 \wedge e_1 \parallel e_2\} .$$

The set E of all switches is again the union of all these sets:

$$E =_{df} F_l \cup G_l \cup H_l .$$

- The jump condition for an edge depends on the set in which the switch is defined. It is given by

$$\begin{aligned} jump(e_1) \wedge jump(e_2) \quad &\text{if} \quad ((^\bullet e_1, {}^\bullet e_2), (e_1^\bullet, e_2^\bullet)) \in F_l , \\ jump(e_1) \quad &\text{if} \quad ((^\bullet e_1, v_2), (e_1^\bullet, v_2)) \in G_l , \\ jump(e_2) \quad &\text{if} \quad ((v_1, {}^\bullet e_2), (v_1, e_2^\bullet)) \in H_l , \end{aligned}$$

where $e_1 \in E_1$, $e_2 \in E_2$, $v_1 \in M_1$ and $v_2 \in M_2$.

As above we can generically create an edge labelling function *event*. Note that the three possibilities for composing hybrid automata coincide if the compatibility relation is chosen as the universal relation, i.e., each switch of H_1 is coupled with every switch of H_2.

Example 2.4.6 Assume again the small and abstract hybrid automata of Example 2.4.4. Using the liberal version of parallel composition yields more switches (cf. Figure 2.8).

In particular, the composed hybrid automaton can now perform at least one transition.

□

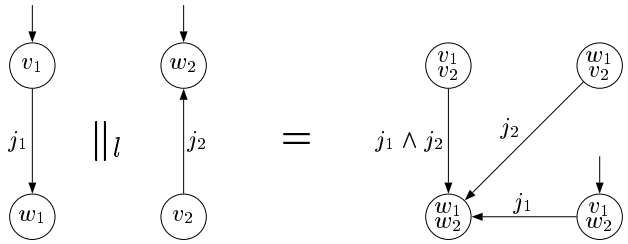


Figure 2.8: Liberal parallel composition of two simple hybrid automata

A real example for two small hybrid systems that are composed in parallel by the above construction is given in the next section.

We now want to discuss how more than two hybrid automata can be composed. For that, we sketch the case of three hybrid automata H_1, H_2 and H_3 and composition relations $\|^{1,2}$ between H_1 and H_2, and $\|^{2,3}$ between H_2 and H_3. More complicated situations (e.g., more involved automata) can handled in a similar way. To determine $(H_1 \|_s H_2) \|_s H_3$ (or $(H_1 \|_l H_2) \|_l H_3$), we only have to define a composition relation $\|^{12,3}$ between $H_1 \|_s H_2$ and H_3. This is straightforward by the above definition $(e_{12}, e_3) \in \|_{s\,12,3} \Leftrightarrow \mathit{event}(e_{12}) = \mathit{event}(e_3)$, for all edges e_{12} of $H_1 \|_s H_2$ and all edges e_3 of H_3.

2.5 Railway Constructions

To round off this chapter, we illustrate the above composition definitions by two further examples. The first one describes a railroad gate control and was described in the context of hybrid automata in [Hen96]. The example was already used before for the synthesis of controllers for real-time discrete event systems [Ost89a]. It will use strict parallel composition (Definition 2.4.3) to build a larger system from smaller ones. In the second example two trains have to share a common single-way track in different directions. To guarantee safety and to build up the whole system we will use the liberal parallel composition (Definition 2.4.5).

A Gate Controller. We assume a circular track between 2000 and 5000 metres long and a railway crossing with a gate. A sketch of the architecture is given in Figure 2.9.

A train travelling on the track is modelled by the hybrid automaton in Figure 2.10.

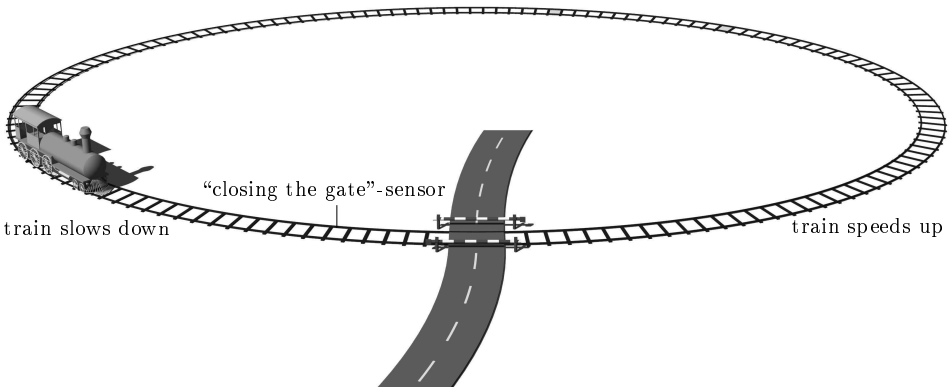


Figure 2.9: Architecture of a simple railroad system [Höf08a]

The variable x represents the distance of the train from the gate. Initially, the speed of the train is between 40 and 50 metres per second. At the distance of 1000 metres from the gate, the train issues an *approach* event and may slow down to 30 metres per second. At the distance of 100 metres behind the gate, the train issues an *exit* event. When this event occurs the distance to the next forthcoming gate has to be set. Depending on the length of the track, x is set to a value between 1900 and 4900.

As a second component of the railroad gate control we have a gate automaton (Figure 2.11).

The variable y of the gate automaton represents the position of the gate in degrees. Initially, the gate is open ($y = 90$). When a *lower* event is received, the gate starts closing at a rate of 9 degrees per second and when a *raise* event happens, the gate starts opening at the same rate.

To combine these two automata one can use a third automaton — a controller automaton — as shown Figure 2.12. This controller has a reaction delay of up to u seconds. For example, if the train issues an *approach* event, the automaton switches to the mode DELAY1. The elapsed time is measured by the variable z. At some point before z reaches the reaction upper bound u the automaton starts the *lower* event and the gate begins to close (the gate automaton is now in mode DOWN).

We build a hybrid system for a gate control from these three hybrid automata using the events *exit*, *approach*, *lower* and *raise*. To construct the hybrid automaton that models the whole system we use strict parallel composition (Definition 2.4.3). The resulting automaton is presented in Appendix C (Figure C.1).

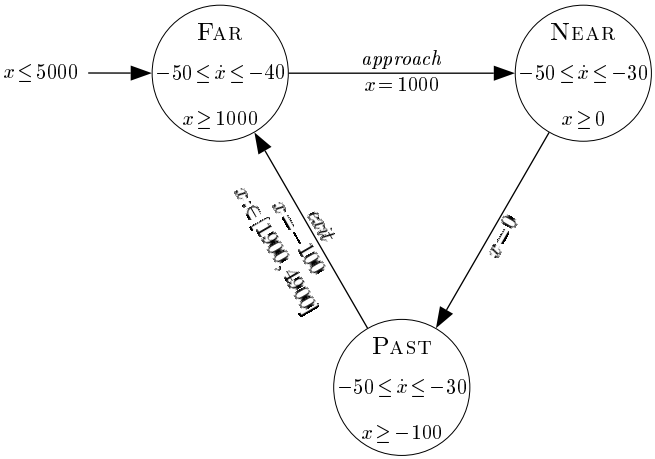


Figure 2.10: Train automaton

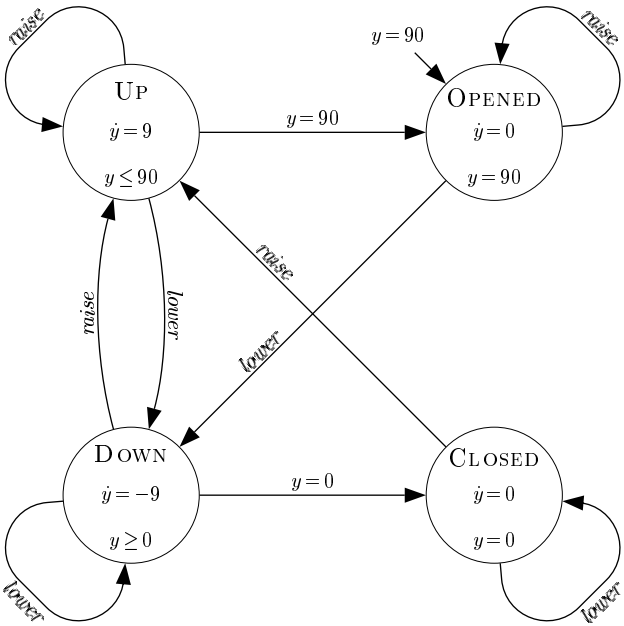


Figure 2.11: Gate automaton

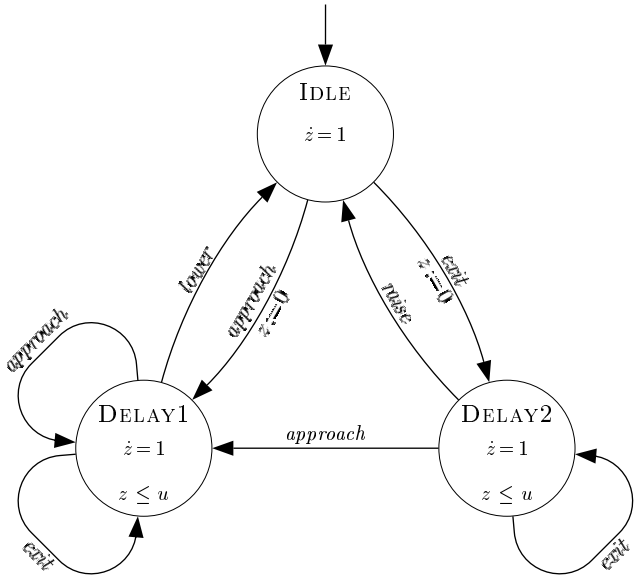


Figure 2.12: Controller automaton

By modularity of hybrid automata, they are adequate for designing and presenting hybrid systems. In our example, we had only three small automata. However, analysing the described systems is not an easy task, especially if the systems interact. Let us have a look at a safety aspect in the example. Can it be guaranteed that the gate is closed at the time when the train passes the gate? For $z = 1$ the answer is for example "yes". But this result is hard to revalidate from the given automata. We think that algebraic expressions are helpful for checking such requirements. We will revisit the example in Chapter 6 after we have developed such an algebra. The use of liberal parallel composition is not useful since in this situation the controller is able to open the gate shortly before the train passes. Hence the above safety criterion is not satisfied for any z. Verifying this behaviour using the composed hybrid automaton (cf. Figure C.2) is again not easy.

A shared single-track way. Now we assume two circular tracks each between 2000 and 5000 metres long. Both tracks share the track at a length of 400 metres. This bottleneck is equipped with a short bypass track that allows two trains passing without colliding. The architecture is given in Figure 2.13.

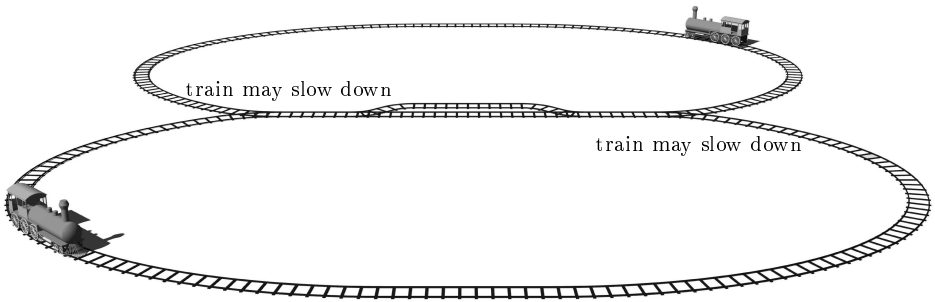


Figure 2.13: Architecture of a railway system with a shared single-track way [Höf08a]

Obviously, this bottleneck in the track needs synchronisation of trains. Each train travelling on the track is modelled by a hybrid automaton (Figure 2.14). The variable x represents the distance of the train between the current position on the track and the beginning of the single-track way. Initially, the speed of the train is between 20 and 40 metres per second. At a distance of 1000 metres from the single track, the train enters an "approaching area" (mode NEAR) where it may slow down or even stop. When reaching the bottleneck it travels with a constant speed of 20 metres per second. After passing the shared track, the distance to the next bottleneck has to

be reset. Depending on the length of the track x is set to a value between 1600 and 4600.

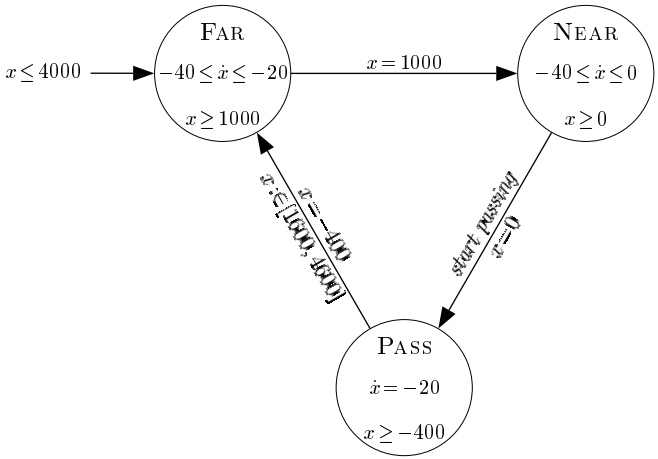


Figure 2.14: Another train automaton

When the two trains are not synchronised by the event *start passing*, they might collide. Using strict parallel composition yields the automaton of Figure 2.15. The variable x represents the behaviour of the first train, y describes the behaviour of the second one. The mode names are the initial letters of the mode names of the single automata. Moreover we have removed unreachable modes.[3] The whole system now guarantees that the trains will not collide at the shared track. It does so at the price of requiring both trains to pass the bottleneck at the same time. As a consequence, a train can pass the bottleneck only if another train is approaching; otherwise it has to stop and wait. This is quite unnatural since synchronisation should only occur if two trains reach the shared track roughly at the same time.

Using the liberal parallel composition yields the correct behaviour. If a train is in mode Near, it checks wether the other train is also approaching. If this is the case, the two trains have to synchronise. If the other train is not approaching (its mode is either Far or Pass) it is allowed to enter the single-track portion (if $x = 0$). The whole automaton is given in Figure C.4.

[3] Unreachable modes are defined either in the sense of finite-state machines, i.e., there exists no path from the initial mode to them, or in the sense of differential equations, i.e., the jump conditions cannot be satisfied at the specific position. The automaton with all possible modes is given in Appendix C.

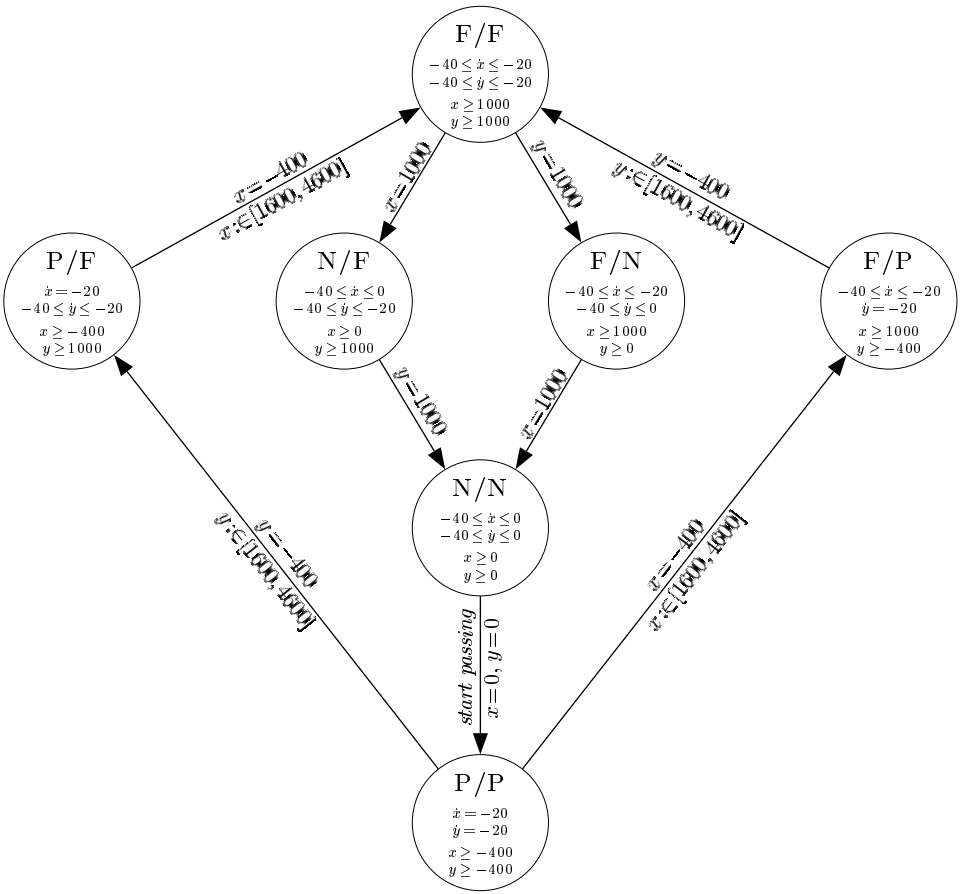


Figure 2.15: Strict parallel composed automaton for a shared single-track way

Chapter 3

Kleene Algebra and its Extensions

Из нета не выкроишь естя.
(Russian proverb)

This chapter recapitulates the notion of *Kleene algebra* which, over the last decades, emerged as a fundamental structure in computing. Such algebras offer a concise syntax for modelling actions, programs or state transitions under non-deterministic choice, sequential composition and finite iteration. Since they provide a uniform semantics for various program analysis tasks, they have found widespread applications ranging from program analysis and semantics to combinatorial optimisation and concurrency control and support cross-theory reasoning with modal, relational, trace-based, language-based and event-based approaches.

Unfortunately, pure Kleene algebras cannot be used to characterise hybrid systems. Therefore we relax the axioms of Kleene algebra and introduce the structures of lazy and weak Kleene algebras. These structures are closely related to R. Dijkstra's computation calculus and von Wright's demonic refinement algebra. Moreover, these structures enable the definition of an algebra of hybrid systems in the next chapter. Extensions for modelling infinite iteration, tests and modal operators are available and will also be recapitulated.

(Lazy/weak) Kleene algebras come with a simple first-order equational calculus that yields particularly short and abstract proofs. Therefore we show how state-of-the-art first-order automated theorem proving (ATP) systems can be used to verify properties within the algebraic setting.

3.1 Kleene Algebras

One of the first uses of Kleene algebra was to axiomatise regular expressions. In particular, they have to model sequential composition, nondeterministic choice and finite iteration. From a mathematical point of view Kleene algebras are based on idempotent semirings. They provide the appropriate level of abstraction for modelling actions, programs or state transitions under non-deterministic choice and sequential composition in a first-order equational calculus.

Definition 3.1.1 ((idempotent) semiring)
A *(full) semiring* is a quintuple $(S, +, \cdot, 0, 1)$ such that $(S, +, 0)$ is a commutative monoid, $(S, \cdot, 1)$ is a monoid, multiplication distributes over addition from the left and right and 0 is a left and right zero of multiplication. This is expressed by the following axioms:

ADDITIVE COMMUTATIVE MONOID

$$a + (b + c) = (a + b) + c \ , \quad (3.1)$$
$$a + 0 = a \ , \quad (3.2)$$
$$a + b = b + a \ , \quad (3.3)$$

MULTIPLICATIVE MONOID

$$a \cdot (b \cdot c) = (a \cdot b) \cdot c \ , \quad (3.4)$$
$$a \cdot 1 = a \ , \quad (3.5)$$
$$1 \cdot a = a \ , \quad (3.6)$$

DISTRIBUTIVITY LAWS

$$(a + b) \cdot c = a \cdot c + b \cdot c \ , \quad (3.7)$$
$$a \cdot (b + c) = a \cdot b + a \cdot c \ , \quad (3.8)$$

LEFT AND RIGHT ANNIHILATION

$$0 \cdot a = 0 \ , \quad (3.9)$$
$$a \cdot 0 = 0 \ . \quad (3.10)$$

As usual we assume that multiplication binds stronger than addition. A semiring S is *idempotent* (an *i-semiring*) if $a + a = a$ for all $a \in S$. On an i-semiring the *natural order* $\leq$ on S is given by $a \leq b \Leftrightarrow_{df} a + b = b$.

The natural order induces an upper semilattice in which $a + b$ is the supremum of a and b and 0 is the least element (e.g., Theorem 6.16 of [HW98, Chapt. I]).

Definition 3.1.2 (quantale)
An i-semiring is called a *quantale* if S is a complete lattice under the natural order and $\cdot$ is universally disjunctive in both arguments.

Following Conway one might call a quantale a *standard Kleene algebra* [Con71].

More interesting behaviours of programs and transition systems arise from finite and infinite iteration.

Definition 3.1.3 (Kleene algebra [Koz94])
A *Kleene algebra* is an i-semiring S extended by an operation $^* : S \to S$ for iterating an element an arbitrary but finite number of times. Such an operation has to satisfy the *star unfold* and the *star induction* axioms

$$1 + a \cdot a^* = a^* , \qquad (3.11) \qquad b + a \cdot c \leq c \Rightarrow a^* \cdot b \leq c , \quad (3.13)$$
$$1 + a^* \cdot a = a^* , \qquad (3.12) \qquad b + c \cdot a \leq c \Rightarrow b \cdot a^* \leq c . \quad (3.14)$$

Note that we could omit either Equation (3.11) or (3.12), since it can be derived from the other axioms (e.g. [Hol98]). Moreover, the two unfold axioms may be given as inequations, since $a^* \leq 1 + a \cdot a^*$ and $a^* \leq 1 + a^* \cdot a$ follows immediately by the induction laws. The expression a^* abstractly represents the reflexive transitive closure of a and is axiomatised in a first-order style. The transitive closure of a is defined as

$$a^+ =_{df} a \cdot a^* \qquad (3.15)$$

To express infinite iteration we axiomatise an omega operator over a Kleene algebra.

Definition 3.1.4 (omega algebra [Coh00])
An *omega algebra* is a Kleene algebra S extended by an operation $^\omega : S \to S$ that satisfies the *omega unfold* and the *omega coinduction* axiom

$$a^\omega = a \cdot a^\omega , \qquad (3.16) \qquad c \leq b + a \cdot c \Rightarrow c \leq a^\omega + a^* \cdot b . \qquad (3.17)$$

Note that we could relax Equation (3.16) to $a^\omega \leq a \cdot a^\omega$, since the converse inequation can be derived immediately. By these definitions, $a^* \cdot b$ and $a^\omega + a^* \cdot b$ are the least and the greatest fixed points of $\lambda x.b + x \cdot a$, respectively if $+$ is completely conjunctive (cf. [ABB+95]). The elements a^* and a^ω arise as special cases.

Idempotent semirings, Kleene algebras and omega algebras have various models and found wide-spread applications. We give some standard and well-known examples. Applications and further information can be found e.g. in [Koz94, Coh94, Koz00, DM01, Str02, Ehm04, DMS04a, DMS04b, Höf05b].

Example 3.1.5
RELATIONS. Consider the structure $\mathsf{REL}(M) =_{df} (\mathcal{P}(M \times M), \cup, ;, \emptyset, Id)$ over an arbitrary set M, where $\mathcal{P}(M \times M)$ is the *set of binary relations* over M, $\cup$ denotes set union, ; stands for the relational product, $\emptyset$ symbolises the empty relation (empty set) and Id denotes the identity relation. It is easy to verify that $\mathsf{REL}(M)$ forms an i-semiring with set inclusion as natural order.

It can be extended to a Kleene algebra by defining R^* as the reflexive transitive closure of an arbitrary relation R, i.e., $R^* =_{df} \bigcup_{i\in\mathbb{N}} R^i$, where, as usual, R^0 is equal to the identity relation and $R^{i+1} = R\,;R^i$.

The structure is also an omega algebra if R^ω is defined as $\nabla R \cdot \top$, where $\top$ denotes the universal relation $M \times M$ and ∇R models those nodes (subidentities) from which R diverges, i.e., from which an infinite R-path emanates.

FORMAL LANGUAGES. One of the oldest and most popular examples is the idempotent semiring of regular *languages*. For some finite alphabet Σ let Σ^* be the set of finite words. We denote the set of all languages by $\mathcal{P}(\Sigma^*)$, the empty word by ε and language concatenation by $L_1.L_2 =_{df} \{v.w \mid v \in L_1,\, w \in L_2\}$, where $v.w$ is the concatenation of the single words v and w. Then the structure $\mathsf{LAN}(\Sigma) =_{df} (\mathcal{P}(\Sigma^*), \cup, ., \emptyset, \{\varepsilon\})$ forms an i-semiring with language inclusion as natural order.

Defining $L^* =_{df} \{w_1.w_2.\cdots.w_n \mid n \in \mathbf{N},\, w_i \in L\}$ turns the structure into a Kleene algebra.

$\mathsf{LAN}(\Sigma)$ also forms an omega algebra, when setting $L^\omega = \Sigma^*$ if $\varepsilon \in L$ and $L^\omega = \emptyset$ otherwise. (see [Ard60, HS08b]). In this case omega does *not* characterise infinite words, since such words are not part of the underlying structure. To allow infinite words and infinite languages one has to relax the structure of Kleene and omega algebra; see Section 3.2.

Next to their classical interpretation as characters, the elements of Σ may e.g. be interpreted as states in a computation system, or, in connection with graph algorithms, as nodes in a graph. So words over Σ can be used to model paths in a transition system.

FINITE TRACES. Besides the model of formal languages we use a second one with a more refined view of multiplication. It uses non-empty words, also called traces, and the *fusion product* $\bowtie_{\mathsf{f}}$ of finite words as a language-valued multiplication operation. For $w_1 \in \Sigma^+$, $w_2 \in \Sigma^+$ and $s \in \Sigma$,

$$w_1 \bowtie_{\mathsf{f}} w_2 =_{df} \begin{cases} \sigma_1.s.\sigma_2 & \text{if } w_1 = \sigma_1.s \text{ and } w_2 = s.\sigma_2 \\ \text{undefined} & \text{otherwise}\,, \end{cases}$$

where $v.w$ denotes again concatenation of words (traces). Informally, a finite non-empty word w_1 can be fused with a non-empty word w_2 if and only if the last letter of w_1 coincides with the first one of w_2; only one copy of that letter is kept in the fused word.

The power-set structure $\mathsf{TRC}(\Sigma) =_{df} (\mathcal{P}(\Sigma^* - \{\varepsilon\}), \cup, \bowtie_{\mathsf{f}}, \emptyset, \Sigma)$ forms an i-semiring if $L_1 \cdot L_2 =_{df} \{w_0 \bowtie_{\mathsf{f}} w_1 \mid\ w_0 \in L_1,\, w_1 \in L_2 \text{ and } w_0 \bowtie_{\mathsf{f}} w_1 \text{ defined}\}$.

Similar to $\mathsf{LAN}(\Sigma)$, it can be extended to a Kleene algebra.

Moreover, $\mathsf{TRC}(\Sigma)$ becomes an omega algebra by setting $L^\omega =_{df} (L_a)^* \cdot L_t \cdot \Sigma^*$, where $L_t = L \cap \Sigma$ and $L_a = L - \Sigma$. But again, this iteration does not cover infinite words [HS08b].

If the underlying set does not matter, we shorten the identifiers $\mathsf{REL}(M)$, $\mathsf{LAN}(\Sigma)$, $\mathsf{TRC}(\Sigma)$ to REL, LAN and TRC, respectively. □

3.2 Relaxing Kleene Algebra

We have seen that Kleene algebras and its variants are powerful structures of the basic control constructs of composition, choice and iteration. Nevertheless, modelling infinite iteration fails in fundamental models. One reason is that the annihilation requirement (Axiom (3.10)) does not allow a natural treatment of lazy computations in which infinite sequences occur [Möl04]. Informally, having an annihilator after an infinite computation does not seem to be reasonable, since the annihilator cannot be reached at all.

Another axiom which should be dropped is the right-distributivity law (Axiom (3.8)). An example where the right-distributivity does not hold are process algebras; Figure 3.1 illustrates the different behaviours.

Figure 3.1: Right-distributivity does not hold in general

On the left hand side, the decision whether process b or c is executed has to be done before process a has started. This decision is reflected by the algebraic formula $a \cdot b + a \cdot c$. On the right hand side, the process a is first executed and afterwards one has to choose between b and c (maybe depending on the result of a). The corresponding formula is $a \cdot (b + c)$. Having these interpretations it is straightforward that the right distributivity law does not hold in general. A more concrete example is given in [MGCM08].

Definition 3.2.1 (lazy/weak semiring)
A *lazy* or *left semiring* is a semiring where the right-annihilation law (Axiom (3.10)) and the right-distributivity law (Axiom (3.8)) are omitted. The lazy semiring is *idempotent* (a *lazy i-semiring*) if $+$ is idempotent and $\cdot$ is right-isotone, i.e., $b \leq c \Rightarrow a \cdot b \leq a \cdot c$, where the *natural order* is given as above. It is *Boolean* if the $\leq$-semilattice is a Boolean algebra with infimum $a \sqcap b$ and complement $\overline{a}$.

A *weak semiring* is a lazy semiring in which $\cdot$ is also right-distributive.

In the sequel, we will consistently write $a, b, \ldots$ for arbitrary lazy semiring elements. Note that left-isotony of $\cdot$ follows immediately from its left-distributivity.

Lemma 3.2.2 *In the setting of lazy i-semirings, right-isotonicity is equivalent to right-super-distributivity, i.e.,* $(b \leq c \Rightarrow a \cdot b \leq a \cdot c) \Leftrightarrow (a \cdot b + a \cdot c \leq a \cdot (b + c))$.

Lemma 3.2.3 *A Boolean lazy semiring satisfies the* shunting rule*, i.e.,*

$$a \sqcap b \leq c \Leftrightarrow b \leq \overline{a} + c \ .$$

In particular, $a \leq b \Leftrightarrow \overline{b} \leq \overline{a}$ *holds.*

Definition 3.2.4 (lazy/weak quantale)
An idempotent lazy/weak semiring S is called a *lazy/weak quantale* if S is a complete lattice under the natural order, $\cdot a$ is universally disjunctive in its left argument and $a \cdot$ is disjunctive if $a \cdot 0 = 0$.

It is straightforward to transfer the definition of Kleene and omega algebra to the lazy setting.

Definition 3.2.5 (lazy/weak Kleene algebra [Möl07])
A *lazy (weak) Kleene algebra* is a structure $(S, {}^*)$ consisting of a lazy (weak) idempotent semiring S and an operation * that satisfies the left unfold and left induction axioms (Equations (3.11) and (3.13)).

Definition 3.2.6 (lazy/weak omega algebra [Möl07])
A *lazy (weak) omega algebra* is a pair $(S, {}^\omega)$ such that S is a lazy (weak) Kleene algebra and an operation $^\omega$ that satisfies Equations (3.16) and (3.17).

Before discussing some basic properties in Section 3.4 we introduce important lazy semirings. Obviously, each full semiring is a lazy and also a weak semiring. But the model class is much richer. Since we are interested in infinite behaviours we are especially interested in semirings where the right annihilation axiom does not hold in general.

Example 3.2.7

OMEGA LANGUAGES. As above (cf. Example 3.1.5), Σ^* is the set of all finite words over an alphabet Σ including the empty word ε. Moreover, Σ^ω is the set of all infinite words over Σ. We set $\Sigma^\infty =_{df} \Sigma^* \cup \Sigma^\omega$. Concatenation of w_1 and w_2 is denoted by $w_1.w_2$, where $w_1.w_2 =_{df} w_1$ if $w_1 \in \Sigma^\omega$.

A *(possible infinite) language* over Σ is a subset of Σ^∞. For a language $L \subseteq \Sigma^\infty$ we define its infinite and finite parts by $\mathsf{inf}\, L =_{df} L \cap \Sigma^\omega$ and $\mathsf{fin}\, L =_{df} L - \mathsf{inf}\, L$.

The lazy Boolean quantale $\mathsf{WOR}(\Sigma) = (\mathcal{P}(\Sigma^\infty), \cup, \emptyset, \cdot, \{\varepsilon\})$ is obtained by extending concatenation from finite to possible infinite languages in the following way:

$$L_1 \cdot L_2 =_{df} \mathsf{inf}\, L_1 \cup (\mathsf{fin}\, L_1).L_2 \ .$$

In general, $L_1.L_2 \neq L_1 \cdot L_2$; if L_2 is the empty language, i.e., $L_2 = \emptyset$, one has $L_1.L_2 = \emptyset$, whereas $L_1 \cdot L_2 = \mathsf{inf}\, L_1$. This algebra is well-known from the classical theory of omega languages (see e.g. [Sta97] for a survey).

TRACES/STREAMS/PATHS. Besides this model we use again a second one with a more refined view of multiplication. Based on the fusion product for finite traces and regarding the cases of infinite words, we set for $w_1, w_2 \in \Sigma^\infty - \varepsilon$ and $s \in \Sigma$

$$w_1 \bowtie w_2 = \begin{cases} \sigma_1.s.\sigma_2 & \text{if } w_1 \in \Sigma^+ \text{ and } w_1 = \sigma_1.s \text{ and } w_2 = s.\sigma_2 \\ w_1 & \text{if } w_1 \in \Sigma^\omega \\ \text{undefined} & \text{otherwise .} \end{cases}$$

As in Example 3.1.5, a finite non-empty word w_1 can be fused with a non-empty word w_2 if and only if the last letter of w_1 coincides with the first one of w_2. If a word is fused with an infinite word from the right, only the infinite word remains as result.

Since we view the infinite words as streams of computations, we call the Boolean lazy quantale $\mathsf{STR}(\Sigma)$ and define it by $\mathsf{STR}(\Sigma) =_{df} (\mathcal{P}(\Sigma^\infty - \varepsilon), \cup, \bowtie, \emptyset, \Sigma)$, where the fusion product $\bowtie$ is extended to languages in the following way:

$$L_1 \bowtie L_2 =_{df} \mathsf{inf}\, L_1 \cup \{w_1 \bowtie w_2 : w_1 \in \mathsf{fin}\, L_1 \wedge w_2 \in L_2\} \ .$$

This operation has the language Σ as its neutral element. Moreover, as above, we have $L_1 \bowtie \emptyset = \mathsf{inf}\, L_1$ and hence $L_1 \bowtie \emptyset = \emptyset$ if and only if $\mathsf{inf}\, L_1 = \emptyset$. A transition relation over a state set Σ can be modelled in $\mathsf{STR}(\Sigma)$ as a set R of words of length 2. The powers R^i of R then consist of the words (or paths) of length $i+1$ that are generated by R-transitions.

The multiplicative identity Σ has exactly the subsets of Σ as its subobjects, so that in this quantale the tests faithfully represent sets of states.

GUARDED STRINGS. Instead of using arbitrary traces, we now want to restrict traces to alternating sequences. A *guarded string* over the sets P and Σ is a possibly infinite word w such that the first letter and, if existent, the last letter are in P and where elements from P and A alternate. Hence the set of possible traces is $(P \times \Sigma)^* \times P \cup (P \times \Sigma)^\omega$. We denote the set of all non-empty guarded strings over P and Σ by $(P, \Sigma)^\infty - \varepsilon$. The structure of sets of alternating sequences $(\mathcal{P}((P, \Sigma)^\infty - \varepsilon), \cup, \bowtie, \emptyset, P)$ with union as addition and the extended fusion product as multiplication forms again a (lazy) semiring. The semiring is full if no infinite sequences are allowed, i.e., the carrier set is restricted to $(P \times \Sigma)^* \times P$; otherwise it is a proper lazy structure. It is called the algebra of *guarded strings* and denoted by $\mathsf{GS}(P, \Sigma)$ By this definition $\mathsf{GS}(P, \Sigma)$ is a subalgebra of $\mathsf{STR}(P \cup \Sigma)$. If no infinite words are allowed the algebra is a subalgebra of $\mathsf{TRC}(P \cup \Sigma)$ and denoted by $\mathsf{FGS}(P, \Sigma)$ (e.g. [Koz97, Koz03]).

Similar to STR, GS can be extended to a lazy Kleene and omega algebra. In this case omega is infinite iteration and behaves naturally, i.e., if $|w| > 1$, then $\{w\}^\omega$ is either empty (at some point the multiplication is not defined) or an infinite word, i.e. $\{w\}^\omega \subseteq (P \times \Sigma)^\omega$.

As in Example 3.1.5, we shorten the identifiers $\mathsf{WOR}(\Sigma)$, $\mathsf{STR}(\Sigma)$, $\mathsf{GS}(P, \Sigma)$ and $\mathsf{FGS}(P, \Sigma)$ to WOR, STR, GS and FGS, respectively, if the underlying sets do not matter. □

It is well known that not every idempotent lazy semiring can be extended to a lazy Kleene algebra. Similarly, not every Kleene algebra forms an omega algebra. One possible argumentation is that every omega algebra has to have a greatest element (cf. Lemma 3.4.4) which is a contradiction to the situation in Kleene algebra.

Lemma 3.2.8

1. *Not every (lazy) i-semiring forms a (lazy) Kleene algebra.*
2. *Not every (lazy) Kleene algebra is a (lazy) omega algebra.*

Proof.

1. The structure $(\mathbb{N} \cup \{-\infty\}, \max, +, -\infty, 0)$ is an idempotent semiring, called the *max-plus semiring* [GM97]. Its natural ordering coincides with the standard ordering on $\mathbb{N} \cup \{-\infty\}$. The max-plus semiring cannot be extended to a Kleene algebra. By simple induction it is easy to verify that $a^n \leq a^*$ for all $n \in \mathbb{N}$. For $a > 0$ the set $\{a^n \mid n \in \mathbb{N}\} = \{n \cdot a \mid n \in \mathbb{N}\}$ is unbounded which contradicts that a^* is an upper bound of a^n.

2. The structure $(\mathbf{N}, \max, \max_0, 0, 1)$ forms an i-semiring if max is defined as usual and $\max_0$ by

$$\max_0(a,b) =_{df} \begin{cases} 0 & \text{if } a = 0 \vee b = 0 \\ \max(a,b) & \text{otherwise .} \end{cases}$$

   The standard ordering on $\mathbf{N}$ coincides with its natural ordering. Since multiplication is idempotent, setting $n^* = \max(1,n)$ turns the structure into a Kleene algebra.

   The verification of the star unfold axiom $1 + n \cdot n^* \leq n^*$ is straightforward by $1 + n \cdot n^* \leq \max(1, n, n^*) = \max(1, n, 1, n) = \max(1, n) = n^*$. We now verify the star induction axiom $l + m \cdot n \leq n \Rightarrow m^* \cdot l \leq n$. If $n \geq 1$, the assumption is $\max(l, m, n) \leq n$, which implies the claim, since $m^* \cdot l = \max(1, m) \cdot l \leq \max(l, 1, m) \leq \max(l, m, n)$. In the case of $n = 0$ the assumption implies $l = 0$ and the claim reduces to $0 \leq 0$. The dual star unfold and star induction axioms follow immediately from commutativity of multiplication.

   Lemma 3.4.4 states that every lazy omega algebra has to have a greatest element. Since $\mathbf{N}$ does not possess such an element the structure is not an omea algebra.

These results transfer directly to the relaxed cases, since every full semiring (Kleene algebra) forms also a lazy one. □

Using the Knaster/Tarski fixpoint theorem (e.g. [DP02]), the existence of operations for finite and infinite iteration can be at least guaranteed for lazy quantales.

Theorem 3.2.9

1. *By defining* $a^* =_{df} \mu x \,.\, a \cdot x + 1$, *every lazy quantale can be extended to a lazy Kleene algebra.*
2. *If the lazy quantale is weak and a completely distributive lattice then it can be extended to a lazy omega algebra by setting* $a^\omega =_{df} \nu x \,.\, a \cdot x$. *In this case,* $\nu x \,.\, a \cdot x + b = a^\omega + a^* \cdot b$.

Proof. We give this proof to pinpoint the use of our assumptions; a similar proof for the more restrictive setting of full quantales appears, e.g., in [ABB+95]. It uses the principles of least and greatest fixpoint fusion (see e.g. [DP02]): Let $f, g, h : L \to L$ be isotone functions on a complete lattice $(L, \leq)$ with least element 0 and greatest element $\top$ such that $g \circ h = f \circ g$.

- If g is continuous, i.e., preserves suprema of nonempty chains, and strict, i.e., satisfies $g(0) = 0$, then $g(\mu h) = \mu f$.

- If g is cocontinuous, i.e., preserves infima of nonempty chains, and costrict, i.e., satisfies $g(\top) = \top$, then $g(\nu h) = \nu f$.

In both parts of the proof we use $f(x) =_{df} a \cdot x + b$, whereas g and h will change.

1. The star axioms (specialised to the case $b = 1$) are equivalent to the statement that a^* is the least contracted element of the function $h(x) =_{df} a \cdot x + 1$; hence by the Knaster/Tarski fixpoint theorem it coincides with the least fixpoint of that function which exists by isotony and completeness assumptions of lazy quantales. Therefore the star unfold axiom holds by construction.

   Now we use least fixpoint fusion with $g(x) =_{df} x \cdot b$ to show that $a^* \cdot b$ is the least fixpoint and hence the least contracted element of f, which is the contents of the star induction axiom.

   By the definition of a lazy quantale, g is continuous and strict. Furthermore,

   $$g(h(x)) \;=\; (a \cdot x + 1) \cdot b \;=\; a \cdot x \cdot b + b \;=\; f(g(x)) \;,$$

   and $a^* \cdot b = \mu f$, as required.

2. The omega unfold axiom holds by construction.
   We set $c =_{df} a^* \cdot b$ and $e =_{df} a^\omega + c$ and show that e is the greatest fixpoint and hence the greatest element expanded by f, which is precisely the contents of the omega coinduction axiom.

   This time we use $g(x) =_{df} x + c$ and $h(x) =_{df} a \cdot x$. Function g is obviously costrict. It is also cocontinuous, since we assume the underlying lazy quantale to be completely distributive. For the commutativity condition we calculate using, first, that c is a fixpoint of f by the proof of Part (1) and, second, weakness of the underlying quantale,

   $$\begin{aligned} g(h(x)) \;&=\; a \cdot x + c \;=\; a \cdot x + f(c) \;=\; a \cdot x + a \cdot c + b \\ &=\; a \cdot (x + c) + b \;=\; a \cdot g(x) + b \;=\; f(g(x)) \;. \end{aligned}$$

   This establishes $\nu f = a^\omega + a^* \cdot b$ as required. □

This theorem implies in particular that every finite Kleene algebra and every set-based Kleene algebra can be extended to an omega algebra. In particular, all structures of Example 3.2.7 form omega algebras. In these algebras star is again finite iteration and omega is indeed infinite iteration.

A useful trick to prove equations with inequations is the following theorem.

Lemma 3.2.10 (indirect equality) *An equation $a = b$ holds if and only if*

$$\forall\, u : u \leq a \;\Leftrightarrow\; u \leq b \qquad \textit{or} \qquad \forall\, u : a \leq u \;\Leftrightarrow\; b \leq u \;.$$

The proof is straightforward and can be automated (see Section 3.3). Before recapitulating useful properties which will be used later on, we will briefly discuss proof automation within the area of Kleene algebra.

3.3 Automation in Kleene Algebra

One advantage of having an algebraic approach is the opportunity of automating proofs. A proof script can not only act as a "certificate", increasing the confidence in the result, but an automated theorem prover tool can fully automate many required facts in a fully formal way.

Not long ago, it was common belief that off-the-shelf automated provers cannot handle structures of comparable complexity. Consequently, implementations of Kleene algebra have already been integrated into interactive higher-order theorem provers [vOG97, Kah04, AHK06, Str02] or special purpose first-order proof systems [MO02]. Nevertheless higher-order theorem provers need a huge amount of user interaction, whereas many first-order provers do not need interaction at all.

Recent experiments show that variants of Kleene algebras can successfully be automated with first-order off-the-shelf technology and applied to various program analysis tasks [HS07, Str07, HS08c, HS08a, HSS09]. An evaluation of various theorem provers shows that Prover9 [McC] and Waldmeister [BHF96, HBV97] are currently best suited for verifying properties in variants of semirings [DH08]. Since Waldmeister can only handle purely equational encodings, we use McCune's Prover9 tool for proving the presented theorems.

Prover9 is a saturation-based theorem prover for first-order equational logic. It implements an ordered resolution and paramodulation calculus and, by its treatment of equality by rewriting rules and Knuth-Bendix completion. Prover9 is complemented by the counterexample generator Mace4, which is very useful in practice. In particular, it can be used to falsify conjectures.

Prover9 and Mace4 accept input in a syntax for first-order equational logic. The input file consists essentially of a list of hypotheses (the set of support), e.g., the axioms of lazy omega algebra, and a goal to be proved. A typical input file is listed in Appendix A.1. Prover9 negates the goal, transforms the hypotheses and the goal into clausal normal form and tries to produce a refutation. Mace4, in contrast, enumerates finite models of the hypotheses and checks whether they are consistent with the goal.

The inference process of saturation-based theorem proving is discussed in detail in the Handbook on Automated Reasoning [RV01]. Roughly, it consists of two interleaved modes.

- The deduction mode closes a given clause set under the inference rules of resolution, factoring and paramodulation. The paramodulation rule implements equational reasoning by replacing equals by equals.
- The simplification mode discards clauses from the working set if they are redundant with respect to other clauses.

In this process, simplification rules are applied eagerly and deduction rules lazily to keep the working set small. The process stops when the closure has been computed or when the empty clause `$F` — which denotes inconsistency — has been produced. Obviously, termination cannot be guaranteed. In the second case, Prover9 reconstructs and displays a proof.

Saturation-based theorem proving implements a semi-decision procedure for first-order equational logic. Whenever the goal is entailed by the hypotheses, the empty clause can be produced in finitely many steps. Otherwise, if the goal is not entailed, a counterexample exists, though not necessarily a finite one.

Since we are interested in robust results that can quickly be obtained by non-experts, we use the prover more or less as a black box and rely on the default strategies provided by Prover9. This makes our experiments more relevant to formal software development contexts.

We have to encode lazy omega algebra for Prover9. This is done in a straightforward way; the code can be found in Appendix A.1. The goal to be proved is also encoded in the same way. For example to prove Lemma 3.2.10 one has to add the lines

```
formulas(goals).
  all a all b (
    a=b <-> all u (u<=a <-> u<=b)
  ).
end_of_list.
```

Here `<=` denotes the natural ordering and `<->` symbolises equivalence. The proof takes around $0.02s$ and is fully automatic. To speed up the proofs one can use hypotheses learning techniques [HS08a, SP07]. This reduces the set of axioms and often yields a proof in less time. Such techniques seem very promising since the simple first-order equational calculus of idempotent lazy semirings (lazy Kleene algebras/lazy omega algebra) yields particularly short proofs.

If possible, we use Prover9 to prove and verify the presented theory. Since Prover9 also provides proofs, we can often omit hand-written proofs. As far as we know this is the first time that automated theorem provers are constistently used in the area of Kleene algebra. The results of the proof automatisation are summarised in

Appendix A. In particular, we indicate which theorems can fully be automated and which need some additional hypotheses. To reproduce the experiments we list all input files at a website [Höf]. Moreover, we present templates for input files in Prover9- and TPTP-syntax in the Appendix. Although the latter one is hard to read by humans, it forms a common basis for development of and experimentation with automated theorem provers; hence every automated theorem prover should accept this format.

3.4 Basic Properties

After the short discussion about automation in Kleene algebra we now recapitulate basic properties of lazy/weak Kleene and omega algebras. All these theorems are proved using Prover9. We only present proofs that are either difficult to automate or include some fundamental ideas.

Most of the presented material of this section are well-known properties of lazy semirings and Kleene algebras. We only recapitulate the basic properties that will be needed afterwards. More rules can be found in the given literature (e.g., [Möl07, Coh00, HM08]). We start with some basic properties for finite and infinite iteration.

Lemma 3.4.1 *Assume a lazy Kleene algebra.*

1. *The inequation* $a \leq a^*$ *holds.*
2. *Star is isotone with respect to the natural order, i.e.,* $a \leq b \Rightarrow a^* \leq b^*$.
3. *Star is idempotent, i.e.,* $a^* \cdot a^* = a^*$ *and* $(a^*)^* = a^*$.
4. $a^* \cdot a \leq a \cdot a^* = a^+$.
5. $(a \cdot b)^* \leq (a + b)^*$.

Although most of the basic laws are similar to laws in full semirings (Kleene algebras), one has to take some care, since the basic laws of lazy semirings are weaker than those for standard ones. However things work out reasonably well and many results come for free. $a \cdot a^*$ is a counterexample; it is not the transitive closure in the general setting of lazy Kleene algebra. This can be shown by the following example, that was immediately found by Mace4.

+	0	1	a	b
0	0	1	a	b
1	1	1	b	b
a	a	b	a	b
b	b	b	b	b

·	0	1	a	b
0	0	0	0	0
1	0	1	a	b
a	0	a	a	b
b	0	b	a	b

*	
0	1
1	1
a	b
b	b

In this case we have $a \cdot a^* = b \neq a = a^* \cdot a$. In particular, $a \cdot a = a$ does not imply $a \cdot a^* = a$. This result can be generalised to the following lemma.

Lemma 3.4.2 *For a weak Kleene algebra the equation $a^* \cdot a = a \cdot a^*$ holds. This implies the second unfold law $1 + a^* \cdot a = a^*$ and $a \cdot a \leq a \Rightarrow a^* \cdot a = a \cdot a^* = a$ which states that a^+ is indeed the transitive closure of a.*

Similar laws as shown in Lemma 3.4.1 hold also for lazy omega algebras.

Lemma 3.4.3 *Assume a lazy omega algebra.*

1. *Omega is isotone with respect to the natural order, i.e., $a \leq b \Rightarrow a^\omega \leq b^\omega$.*
2. *Omega is weak idempotent, i.e., $a^\omega \cdot a^\omega \leq a^\omega$ and $(a^\omega)^\omega \leq a^\omega$.*
3. *The identity $a^* \cdot a^\omega = a^\omega$ holds.*
4. *If a is idempotent, i.e., $a \cdot a = a$, then $a^\omega = a \cdot \top$.*
5. *The omega coinduction (3.17) implies $c \leq a \cdot c \Rightarrow c \leq a^\omega$.*

Lemma 3.4.4 *Each omega algebra has a greatest element $\top =_{df} 1^\omega$. Furthermore, a^ω is a left ideal, i.e., $a^\omega = a^\omega \cdot \top$.*

The following lemma finishes the list of properties for lazy omega algebras which are needed. Some of them can be found in [Möl07].

Lemma 3.4.5 *Assume a lazy omega algebra.*

1. $a \cdot (b \cdot a)^\omega = (a \cdot b)^\omega$.
2. $a^\omega \cdot b \leq a^\omega$.
3. $(a \cdot b)^\omega \leq (a + b)^\omega$.
4. $\forall i \in \mathbb{N}, i > 0 : (a^i)^\omega \leq (a^+)^\omega = a^\omega$.
5. $(a + b)^\omega = (a^* \cdot b)^\omega + (a^* \cdot b)^* \cdot a^\omega$.
6. $(a + b)^\omega = a^\omega + (a^* \cdot b) \cdot (a + b)^\omega$.

Again all proofs (except the first inequality of Lemma 3.4.5(4)) have been done by the automated theorem prover Prover9. The property $(a^i)^\omega \leq (a^+)^\omega$ cannot be encoded with Prover9 because it is universally quantified. But it is a simple consequence of $a^i \leq a^+$ and isotony.

When defining an algebra of hybrid systems in Section 4.1, we will distinguish between purely finite and purely infinite sets of trajectories. Informally, a set of trajectories

is purely finite if its elements are finite. A set is purely infinite if it contains only trajectories with infinite length. In [Möl07] Möller shows a possibility to distinguish between purely finite and purely infinite elements in the general case of lazy semirings. We recapitulate again only the necessary theory needed later on. For further details see the original work.

Definition 3.4.6 (purely infinite parts/elements)
For an idempotent lazy semiring, the *purely infinite part* of an element a is defined as $\mathsf{inf}\, a =_{df} a \cdot 0$ and a is called *purely infinite* if $a \cdot 0 = a$.

This property is equivalent to a being a *left zero*, i.e., to $\forall\, b : a \cdot b = a$.

Often there is a largest purely infinite element N characterised by $a \leq \mathsf{N} \Leftrightarrow a \cdot 0 = a$.

Lemma 3.4.7 *If there exists a largest purely infinite element* N *in a lazy semiring, then for all* a*,*

$$\mathsf{N} \cdot a \leq \mathsf{N} \quad \textit{and} \quad a \cdot \mathsf{N} \leq \mathsf{N} \ .$$

Informally this lemma states that an infinite element cannot become finite when composing with an arbitrary element.

Dually to Definition 3.4.6, we can characterise purely finite parts.

Definition 3.4.8 (purely finite parts/elements)
For a lazy semiring, an element a is *purely finite* if its purely infinite part is trivial, i.e., if $\mathsf{inf}\, a = a \cdot 0 = 0$.

In many semirings there exists next to N a largest purely finite element F characterised by $a \leq \mathsf{F} \Leftrightarrow a \cdot 0 = 0$.

Lemma 3.4.9 *If a lazy semiring contains a largest purely finite element* F*, then*

$$\mathsf{F} \cdot \mathsf{F} = \mathsf{F} \ .$$

Theorem 3.4.10 (existence of greatest purely (in)finite elements)
In Boolean lazy semirings N *and* F *always exist and satisfy*

$$\mathsf{N} = \top \cdot 0 \quad \textit{and} \quad \mathsf{F} = \overline{\mathsf{N}} \ ,$$

where $\top =_{df} \overline{0}$ *denotes the greatest element.*

The proof is immediate by the definitions of $\top$, N and F. This implies that in the case of Boolean lazy semirings every element can be split into its purely finite and its purely infinite parts, i.e., $a = \mathsf{fin}\, a + \mathsf{inf}\, a$, where $\mathsf{fin}\, a = a \sqcap \mathsf{F}$ and $\mathsf{inf}\, a = a \sqcap \mathsf{N}$. In particular, these parts are disjoint. This motivates the following definition.

Definition 3.4.11 (separated lazy semiring)
An idempotent lazy semiring S is called *separated* if every element $a \in S$ can be split into its purely finite and purely infinite parts. This can be expressed as follows:

$$\mathsf{inf}\, a = a \cdot 0 \;, \qquad (3.18) \qquad\qquad a = \mathsf{fin}\, a + \mathsf{inf}\, a \;, \qquad (3.21)$$

$$\mathsf{fin}\,(\mathsf{inf}\, a) = 0 \;, \qquad (3.19) \qquad\qquad \mathsf{fin}\,(a+b) = \mathsf{fin}\, a + \mathsf{fin}\, b \;, \qquad (3.22)$$

$$\mathsf{inf}\,(\mathsf{fin}\, a) = 0 \;, \qquad (3.20) \qquad\qquad \mathsf{inf}\,(a+b) = \mathsf{inf}\, a + \mathsf{inf}\, b \;, \qquad (3.23)$$

Equation (3.18) defines the infinite part of an element as before; Equations (3.19) and (3.20) enforce that there is no purely infinite element with purely finite parts and vice versa; that means that the set of purely infinite and purely finite elements are disjoint; (3.21) characterises that each part of an element is either purely finite of infinite. Finally, the last two equations enforce additivity of both operators. In particular, these both equations imply immediately $\mathsf{fin}\,(\sum_{i\in I} a_i) = \sum_{i\in I} \mathsf{fin}\, a_i$ and $\mathsf{inf}\,(\sum_{i\in I} a_i) = \sum_{i\in I} \mathsf{inf}\, a_i$ if I is finite. The Axioms (3.18)–(3.22) are irredundant, i.e., no of these axioms is entailed by the lazy semiring axioms and the remaining axioms for separation. This can be shown by the counterexample generator Mace4 [McC]. Axiom (3.23) is an exception; it follows directly from Axiom (3.18) and right-distributivity. However we add this axiom due to symmetry.

Example 3.4.12 In [Möl04], Möller shows that every Boolean lazy i-semiring is separated. This implies that $\mathsf{WOR}(\Sigma)$ and $\mathsf{STR}(\Sigma)$ are separated. Moreover the operators fin and inf behave naturally. Assume an element L from $\mathsf{WOR}(\Sigma)$ or $\mathsf{STR}(\Sigma)$. Then

$$\mathsf{fin}\, L = \{w \mid w \in L, |w| < \infty\} \qquad \text{and} \qquad \mathsf{inf}\, L = \{w \mid w \in L, |w| = \infty\} \;,$$

where $|w|$ denotes the length of w. □

Lemma 3.4.13 *For a separated lazy semiring the operation* inf /fin *commute with* $\sqcap$ *(if* $\sqcap$ *exists).*

$$(\mathsf{inf}\, a) \;\sqcap\; b = \mathsf{inf}\,(a \sqcap b) \quad \textit{and} \quad (\mathsf{fin}\, a) \;\sqcap\; b = \mathsf{fin}\,(a \sqcap b) \;.$$

Lemma 3.4.14 *For arbitrary elements a and b of a separated lazy semiring, the purely finite and purely infinite parts of a composition satisfy the following laws:*

1. $a \cdot b = \mathsf{inf}\, a + (\mathsf{fin}\, a) \cdot b$.
2. $\mathsf{inf}\,(a \cdot b) = \mathsf{inf}\, a + \mathsf{fin}\, a \cdot \mathsf{inf}\, b$.
3. $\mathsf{fin}\,(a \cdot b) = \mathsf{fin}\,(\mathsf{fin}\, a \cdot b) \;\geq\; \mathsf{fin}\, a \cdot \mathsf{fin}\, b$.
4. *If S is weak, the inequation of Part (3) strengthens to an equality.*

We now state further general laws concerning purely finite and purely infinite parts.

Lemma 3.4.15 *Let S be a separated lazy semiring, Kleene algebra and Omega algebra, respectively*

1. $a \leq \mathsf{F} \Leftrightarrow a = \mathsf{fin}\, a \Leftrightarrow \mathsf{inf}\, a = 0$ *and*
   $a \leq \mathsf{N} \Leftrightarrow a = \mathsf{inf}\, a \Leftrightarrow \mathsf{fin}\, a = 0$.
2. *For* $a, b \leq \mathsf{F}$ *and* $c, d \leq \mathsf{N}$ *we have* $a + c \leq b + d \Leftrightarrow a \leq b \wedge c \leq d$.
3. $a^\omega = (\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + (\mathsf{fin}\, a)^\omega$.
4. $\mathsf{inf}\, a^\omega = (\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + \mathsf{inf}\, ((\mathsf{fin}\, a)^\omega)$.
5. *If S is even Boolean, then* $\mathsf{fin}\, a^\omega = (\mathsf{fin}\, a)^\omega \sqcap \mathsf{F} \leq (\mathsf{fin}\, a)^\omega$.

Part (1) gives equivalent characterisations of purely finite and purely infinite elements which are calculationally useful in various circumstances. Part (2) means that a sum of a purely finite and a purely infinite element can uniquely be decomposed again. Part (3) says that infinite iteration of a can take two forms: it may proceed a while with using finite parts of a, but then followed by an element of the infinite part which prohibits further iteration — or it keeps iterating finite parts forever. The meaning of the remaining parts will be discussed after we have introduced the algebra of hybrid systems in Section 4.1.

3.5 Tests

To model assertions and guards in the setting of (lazy) semirings we use the algebraic concept of tests. Tests were introduced into semirings by Manes and Benson [MB85] and into Kleene algebras by Kozen [Koz97].

Definition 3.5.1 (test)
A *test* in an idempotent lazy semiring (quantale) is an element $p \leq 1$ that has a complement q relative to 1, i.e., $p + q = 1$ and $p \cdot q = 0 = q \cdot p$. The set of all tests of S is denoted by $\mathsf{test}(S)$.

In the sequel, we will consistently write $p, q, \ldots$ for arbitrary test elements.

It is not hard to show that the complement $\neg p$ of a test p is uniquely determined by the definition and that in a weak semiring $\mathsf{test}(S)$ is closed under $+$ and $\cdot$ and forms a Boolean algebra with 0 and 1 as its least and greatest elements. To establish this in general lazy semirings one has to add the assumption $p \cdot (q + r) = p \cdot q + p \cdot r$ of right-distributivity of tests among each other.

Since $\mathsf{test}(S)$ forms a Boolean algebra, we have similar shunting rules for tests like we have for Boolean semirings (Lemma 3.2.3).

Lemma 3.5.2 (test shunting) *For tests p, q, r the equation*

$$p \cdot q \leq r \;\Leftrightarrow\; p \leq \neg q + r$$

holds. This implies immediately $p \leq q \;\Leftrightarrow\; \neg q \leq \neg p$.

Moreover, by this lemma and $p \leq 1$, all tests are purely finite. If S itself is Boolean, then $\mathsf{test}(S)$ coincides with the set of all elements below 1.

With the above definition of tests we deviate slightly from [Koz97], where an arbitrary Boolean algebra of subidentities is allowed as $\mathsf{test}(S)$. The reason is that, as shown in Theorem 4.15 of [DMS06], the axiomatisation of domain to be presented below forces every complemented subidentity to be in $\mathsf{test}(S)$ anyway.

An important property of lazy semirings is distribution of test multiplication over meet [Möl07].

Lemma 3.5.3 (distribution of test multiplication over meet) *If the meet $a \sqcap b$ exists for a lazy semiring S (e.g., if S is Boolean) then so do the meets $p \cdot a \;\sqcap\; b$ and $p \cdot a \;\sqcap\; p \cdot b$ and satisfy*

$$p \cdot (a \sqcap b) \;=\; p \cdot a \;\sqcap\; b \;=\; p \cdot a \;\sqcap\; p \cdot b \;.$$

As a direct consequence, we get $p \cdot \top \sqcap a = p \cdot a$ and $p \cdot a \sqcap q \cdot a = p \cdot q \cdot a$. If S is right-distributive, also the symmetric properties hold.

Corollary 3.5.4 *If S is Boolean, the relationship $\overline{p \cdot a} = \neg p \cdot a + \overline{a}$ holds. In particular, $\overline{p} = \neg p + \overline{1}$ and $\neg p \cdot \top = \overline{p \cdot \top}$. If S is right-distributive and separated we get $\overline{a \cdot p} = \mathsf{fin}\,(a) \cdot \neg p + \overline{a}$ and hence $\top \cdot p = \overline{\mathsf{F} \cdot \neg p}$ holds.*

This shows again that the introduced structures are not symmetric and need special care when calculating.

As before, we list a number of important properties which will be needed. All proofs are automated; hand-written proofs can be found in [Möl07].

Lemma 3.5.5 *Assume a lazy i-semiring S with tests.*

1. *If a greatest element exists, then $p \leq q \;\Leftrightarrow\; p \cdot \top \leq q \cdot \top$.*
2. *If S is also a lazy Kleene algebra or omega algebra, then $p^* = 1$ and $p^\omega = p \cdot \top$.*
3. *If S is a lazy omega algebra, then $p \cdot (p \cdot a)^\omega = (p \cdot a)^\omega$ and $(p \cdot a)^\omega = (p \cdot a \cdot p)^\omega$.*

By Part (1) the set of test ideals is isomorphic to the set of tests.

Kleene algebra with tests were first introduced as an equational system for manipulating programs [Koz97]. It was Kozen who first give an algebraic and purely equational proof using such a structure for the following classical result: "every while program can be simulated by a while program with at most one while loop." Afterwards tests have been successfully applied to basic safety analysis, source-to-source program transformation, and concurrency control. It has also been shown that Kleene algebra with tests subsumes propositional Hoare logic [Koz00]. Tests will play an important rôle in the description of hybrid systems, as we will see in the next chapters.

3.6 Domain and Codomain

Domain and codomain are intended to abstractly characterise, in the form of tests, the sets of initial and final states of a set of computations. In combination with restriction, domain yields an abstract preimage operation and codomain an abstract image operation. Desharnais, Möller and Struth extended Kleene algebra by domain and codomain operators which links algebraic, relational and modal approaches; it provides equal opportunities for propositions and actions [DMS06]. In contrast to the domain and codomain operators of full semirings the operators for lazy semirings are not symmetric. Again we recapitulate their definitions and establish some properties which we need afterwards.

Definition 3.6.1 (lazy semiring with domain [Möl07])
A *lazy semiring with domain (quantale)* is a structure $(S, \ulcorner)$, where S is an idempotent lazy test semiring and the *domain operation* $\ulcorner\colon S \to \mathsf{test}(S)$ satisfies for all $a, b \in S$ and $p \in \mathsf{test}(S)$

$$a \leq \ulcorner a \cdot a\ , \quad \text{(d1)} \qquad \ulcorner(p \cdot a) \leq p\ , \quad \text{(d2)} \qquad \ulcorner(a \cdot \ulcorner b) \leq \ulcorner(a \cdot b)\ . \quad \text{(d3)}$$

Sometimes we call such a structure also a domain lazy semiring.

Obviously the same definition work for weak and full semirings as well as for Kleene and omega algebras.

In the literature concerning full semirings with domain, only the Axioms (d1) and (d2) are postulated. If the third axiom (d3) is satisfied a semiring with domain is called modal. However, in the case of lazy semirings one usually postulates this axiom from the beginning.

The axioms are the same as in [DMS06]. Contrarily to the case of arbitrary semirings [Möl05a] with complete sublattice of tests, the domain operation is guaranteed

to exist in lazy quantales. Since the domain operator describes all possible starting states of an element, it is easy to see that "laziness" of the underlying semiring does not matter. Their relevant consequences can still be proved over lazy semirings (quantales) [Möl07]. Examples are summarised in the following lemmas.

Lemma 3.6.2 *The conjunction of (d1) and (d2) is equivalent to each of*

$$\lceil a \leq p \Leftrightarrow a \leq p \cdot a \ , \quad \text{(llp)} \qquad\qquad \lceil a \leq p \Leftrightarrow \neg p \cdot a \leq 0 \ . \quad \text{(gla)}$$

Property (llp) says that $\lceil a$ is the least left preserver of a; (gla) that $\neg\lceil a$ is the greatest left annihilator of a. By Boolean algebra, (gla) is equivalent to $p \cdot \lceil a \leq 0 \Leftrightarrow p \cdot a \leq 0$.

Lemma 3.6.3 *Assume a lazy semiring S with domain.*

1. *Domain is isotone with respect to the natural order.*
2. *Domain is universally disjunctive. In particular, $\lceil 0 = 0$ and $\lceil(a + b) = \lceil a + \lceil b$. If S is Boolean, then $\lceil a + \lceil \overline{a} = 1$.*
3. *Domain is fully strict, i.e., $\lceil a \leq 0 \Leftrightarrow a \leq 0$.*
4. *Domain of tests is stable, i.e., $\lceil p = p$. If there is a greatest element, then the equation $\lceil(p \cdot \top) = p$ also holds.*
5. *The import/export rule $\lceil(p \cdot a) = p \cdot \lceil a$ is satisfied.*
6. *Furthermore , $\lceil(a \cdot b) \leq \lceil a$ holds.*

The first statement of Part (2) cannot encoded in first-order style. However the proof is straightforward and a simple exercise.

Over the Boolean lazy quantale $\mathsf{TRC}(\Sigma)$ the domain operation $\lceil\_ : S \to \mathsf{test}(S)$ returns, for a set of paths represented by an element from $\mathsf{TRC}(\Sigma)$, the set of their starting states. In this case it can also be axiomatised by a Galois connection.

Lemma 3.6.4 *Assume a Boolean lazy quantale with greatest element $\top$. Then*

$$\lceil a \leq p \Leftrightarrow a \leq p \cdot \top \ .$$

We now turn to the dual case of the domain operation. In the case where we have (as in full semirings) right-distributivity and right-strictness, a codomain operation $\rceil$ is easily defined as a domain operation in the opposite lazy semiring (i.e., the one that swaps the order of composition). But due to the absence of right-distributivity and right-strictness we need an additional axiom.

Definition 3.6.5 (lazy semiring with codomain [Möl07])
A *lazy semiring with codomain* is a structure $(S, {}^{\rceil})$, where S is an idempotent lazy test semiring and the *codomain operation* ${}^{\rceil} : S \to \mathsf{test}(S)$ satisfies for all $a, b \in S$ and $p \in \mathsf{test}(S)$

$$a \leq a \cdot a^{\rceil} \ , \quad \text{(cd1)} \qquad (a \cdot p)^{\rceil} \leq p \ , \quad \text{(cd2)} \qquad (a^{\rceil} \cdot b)^{\rceil} \leq (a \cdot b)^{\rceil} \ , \quad \text{(cd3)}$$

$$(a + b)^{\rceil} \geq a^{\rceil} + b^{\rceil} \ . \quad \text{(cd4)}$$

The additional axiom (cd4) guarantees isotony of the codomain operator. As for domain, $a^{\rceil}$ is the least right preserver of a. However, due to lack of right-strictness $\neg a^{\rceil}$ need not be the greatest right annihilator; we only have a weaker equivalence.

Lemma 3.6.6 *The conjunction of (cd1) and (cd2) is equivalent to*

$$a^{\rceil} \leq p \Leftrightarrow a \leq a \cdot p \ , \quad \text{(lrp)} \qquad a^{\rceil} \leq p \Leftrightarrow a \cdot \neg p \leq a \cdot 0 \ . \quad \text{(wgra)}$$

In the general setting of lazy semirings, the (wgra) does not imply (cd1) and (cd2).

Similar to Lemma 3.6.3, we can prove some basic properties for codomain.

Lemma 3.6.7 *Assume a lazy semiring S with codomain.*

1. *Codomain is isotone with respect to the natural order.*
2. *Codomain is universally disjunctive. In particular $0^{\rceil} = 0$ and $(a + b)^{\rceil} = a^{\rceil} + b^{\rceil}$. If S is Boolean, then $a^{\rceil} + \overline{a}^{\rceil} = 1$.*
3. *Codomain is not strict, but $a^{\rceil} \leq 0 \Leftrightarrow a \leq \mathsf{N}$ holds.*
4. *Codomain of test is stable, i.e., $p^{\rceil} = p$.*
5. *The import/export rule $(a \cdot p)^{\rceil} = a^{\rceil} \cdot p$ is satisfied.*
6. *Furthermore, $(a \cdot b)^{\rceil} \leq b^{\rceil}$ holds.*

Lemma 3.6.3(3) and Lemma 3.6.7(3) show the asymmetry of domain and codomain.
Following the lines of [Möl07], we define a modal lazy semiring as a lazy semiring where domain and codomain exist.

Definition 3.6.8 (modal lazy semiring)
A *modal lazy semiring* is an lazy semiring with domain and codomain.

Example 3.6.9 Both $\mathsf{WOR}(\Sigma)$ and $\mathsf{STR}(\Sigma)$ are modal. □

The following lemma has some important consequences for the following chapters, and illustrates again the asymmetry of lazy semiring.

Lemma 3.6.10 *Assume a lazy semiring with (co)domain and a greatest element* $\top$. *Then*

1. $\neg p \cdot a \leq 0 \Leftrightarrow \lceil a \leq p \Leftrightarrow a \leq p \cdot a \Leftrightarrow a \leq p \cdot \top$.
2. $a \cdot \neg p \leq a \cdot 0 \Leftrightarrow a\rceil \leq p \Leftrightarrow a \leq a \cdot p \Leftrightarrow a \leq \top \cdot p$.
3. $a \leq \mathsf{F} \Leftrightarrow (a \leq a \cdot p \Leftrightarrow a \cdot \neg p \leq 0) \Leftrightarrow (a \leq \top \cdot p \Leftrightarrow a \cdot \neg p \leq 0)$.
   Therefore, in general, $a \leq a \cdot p \not\Rightarrow a \cdot \neg p \leq 0$ *and* $a \leq \top \cdot p \not\Rightarrow a \cdot \neg p \leq 0$.

Proof.

1. The first equivalence is (gla), the second (llp). $a \leq p \cdot a \Rightarrow a \leq p \cdot \top$ holds by isotony of $\cdot$ and $a \leq p \cdot \top \Rightarrow \lceil a \leq p$ by isotony of domain and $\lceil(p \cdot \top) = p$ (Lemma 3.6.3(4)).
2. Symmetrically to (1).
3. $a \leq \mathsf{F} \Rightarrow (a \leq a \cdot p \Leftrightarrow a \cdot \neg p \leq 0)$ holds by (2) and $a \cdot 0 \leq 0 \Leftrightarrow a \leq \mathsf{F}$.
   The converse implication is shown by setting $p = 1$, Boolean algebra and definition of F: $a \leq a \Rightarrow a \cdot \neg 1 \leq 0 \Leftrightarrow a \cdot 0 \leq 0 \Leftrightarrow a \leq \mathsf{F}$.
   The second equivalence follows from $a \leq a \cdot p \Leftrightarrow a \leq \top \cdot p$ (see Part (2)). □

Part (3) says that we do not have a law for codomain that is symmetric to (1). Further properties of (co)domain and modal lazy semirings can be found in [DMS06, Möl07].

Using a subalgebra as test algebra yields a two-sorted approach for tests and domain which seems to be quite unnatural. Most recently, Desharnais and Struth showed that the domain operator for full semirings can be generalised and therefore the underlying Boolean algebra does not need an extra sort [DS08]. That means that the (co)domain operator can be defined without using the test subalgebra. They proved that a semiring S is a full semiring satisfying (d1)–(d3) if and only if it can be extended by an *antidomain* operation $\partial : S \to S$ that satisfies the axioms

$$\partial(a) \cdot a = 0 \,, \qquad \partial(a \cdot b) \leq \partial(a \cdot \partial(\partial(b))) \,, \qquad \partial(\partial(a)) + \partial(a) = 1 \,.$$

The correspondence between antidomain, domain and test is then given by $\lceil a = \partial(\partial(a))$, $\neg\lceil a = \partial(a)$ and $\mathsf{test}(S) = \{\partial a \,|\, a \in S\}$. However, since we will need tests anyway and since it has not be analysed if their work can be translated to lazy semirings[4], we use the "classical" definition of domain and codomain using tests.

The three simple equational domain axioms or the antidomain function allow the definition of (forward) modal operators semantically via abstract image and preimage operations [DMS04a, MS06].

[4] We do not expect any problems.

Definition 3.6.11 (modal operators)
Let S be a modal lazy semiring and $p \in \mathsf{test}(S)$. The *(forward) diamond operator* is defined via abstract preimage. The *(forward) box operator* is, as usual, the de Morgan dual.

$$|a\rangle q =_{df} \ulcorner(a \cdot q) \ , \qquad |a]q =_{df} \neg|a\rangle\neg q \ .$$

These operators will occur in the algebraic frameworks for hybrid programs in Chapter 4; hence they should be mentioned. In many cases, expressions that mention modalities can be reduced to pure Kleene algebra with tests. This preserves the algorithmic complexity and provides a very symmetric approach to reasoning about actions and propositions or transitions and states. The diamond is an abstract inverse-image operator, whereas box generalises the notion of the weakest liberal precondition wlp to Boolean lazy quantales. If we view a as the transition relation of a command then the test $|a]q$ characterises those states from which no transition under a is possible or the execution of a is guaranteed to end up in a final state that satisfies test q. Both operators are isotone in their test argument. Hence in a Boolean quantale we have the full power of the modal μ-calculus [HKT00] available.

One can define also backward diamond and box operators using codomain. However since we will not use them we skip them here.

Properties about modal operators can be found in [DMS04a, MS06].

Chapter 4

Algebra of Hybrid Systems

[...] De Manera, que quien sabe por Algebra,
sabe scientificamente.
P. Nuñes (Libro de Algebra en Arithmetica y Geometria)

In this chapter we combine the concepts presented before. In particular, we introduce an algebra of hybrid systems. Elements of such an algebra are sets of trajectories. We show that the structure forms a weak Kleene algebra. This algebra paves the way to use off-the-shelf theorem prover within the area of hybrid systems. We present algebraic properties, including a characterisation of Zeno effects. Furthermore, we show how hybrid system composition can be modelled algebraically and how safety and liveness properties can be formalised.

4.1 Trajectory-Based Model

As mentioned in Chapter 2, trajectories reflect the variation of the values of the variables over time $\mathbb{R}_{\geq 0}$. We abstract from $\mathbb{R}_{\geq 0}$ to a generalise time domain. Before redefining trajectories in this setting we need a couple of definitions.

Definition 4.1.1 (time domain)
A *time domain* is a set of *durations* equipped with a cancellative addition operator $+$ and an element $0 \in D$ such that $(D, +, 0)$ is a commutative monoid. Furthermore, the relation $d_1 \leq d_2 \Leftrightarrow_{df} \exists\, d : d_1 + d = d_2$ has to be a linear order on D.

In the definition cancellativity means that for arbitrary elements d_1, d_2 and d_3 the implication $d_1 + d_3 = d_2 + d_3 \Rightarrow d_1 = d_2$ holds. Examples for time domains are $\mathbb{N}$, $\mathbb{Q}_{\geq 0}$ and $\mathbb{R}_{\geq 0}$.

Lemma 4.1.2 *In a time domain D, 0 is the least element and $+$ is isotone with respect to $\leq$. Moreover, 0 is indivisible, i.e., $d_1 + d_2 = 0 \Leftrightarrow d_1 = d_2 = 0$.*

D may include a special value describing an infinite duration.

Definition 4.1.3 (infinite duration)
A time domain D contains an element ∞ for *infinite duration* iff ∞ is an annihilator with respect to $+$ and cancellativity of $+$ is restricted to all other elements $(D - \{\infty\})$. A time domain is called *infinite* if it includes ∞.

Lemma 4.1.4 *For an infinite time domain D, ∞ is the greatest element of D. In particular, ∞ is unique.*

Definition 4.1.5 (interval)
For an arbitrary duration d of a time domain D, we define the *interval* $\mathsf{intv}\, d$ of admissible times as

$$\mathsf{intv}\, d =_{df} \begin{cases} [0, d] & \text{if } d \neq \infty \\ [0, d[& \text{otherwise .} \end{cases}$$

Using the definition for intervals we can now characterise trajectories over a general time domain.

Definition 4.1.6 (trajectory, duration, range)
Let V be a set of *values* and D a (infinite) time domain. A *trajectory* τ is a pair (d, g), where $d \in D$ and $g : \mathsf{intv}\, d \to V$. d is called the *duration* of the trajectory and the image of $\mathsf{intv}\, d$ under g is its *range* $\mathsf{ran}\ (d, g)$.

A special rôle is played by *zero-duration trajectories* of the form $\underline{x} =_{df} (0, g)$ with $x \in V$ and $g(0) =_{df} x$; they represent single values of the system. Hence they will be used to test wether a value holds within a trajectory or not.

Looking at the above definition we see that all trajectories start at time point 0. This normalisation abstracts from the problem of having various starting points, but needs a sophisticated definition of composition. Figure 4.1 illustrates the main idea. A composed trajectory $\tau_1 \cdot \tau_2$ follows first τ_1. If the end of τ_1 is reached, it starts following τ_2. There are only two questions that have to be considered. First, what happens if the duration of τ_1 is infinite? Second, do we need any restriction at the composition point?

The answer to the first question is rather simple. If τ_1 is infinite, the τ_2 will never reached, hence the composed trajectory $\tau_1 \cdot \tau_2$ equals τ_1. The answer for the second question is more complicated. For the moment we will assume that trajectories can only composed if the values of τ_1 and τ_2 coincide at the composition point. This yields the following definition.

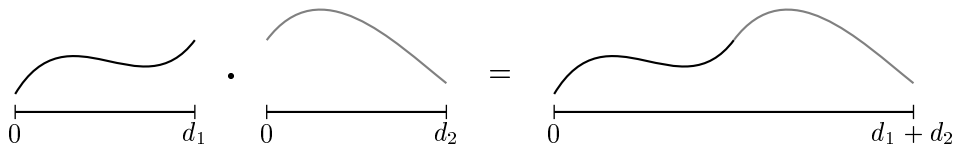


Figure 4.1: Composition of two finite trajectories

Definition 4.1.7 (composition of trajectories)
The *composition of trajectories* (d_1, g_1) and (d_2, g_2) is defined as

$$(d_1, g_1) \cdot (d_2, g_2) =_{df} \begin{cases} (d_1 + d_2, g) & \text{if } d_1 \neq \infty \wedge g_1(d_1) = g_2(0) \\ (d_1, g_1) & \text{if } d_1 = \infty \\ \text{undefined} & \text{otherwise} \end{cases}$$

with $g(t) = g_1(t)$ for all $t \in [0, d_1]$ and $g(t + d_1) = g_2(x)$ for all $t \in \mathsf{intv}\, d_2$.

Composition of trajectories is well defined by cancellativity of $+$ on durations other than ∞. Moreover this operation is associative. The condition $g_1(d_1) = g_2(0)$ for composing trajectories is sometimes too restrictive. In Section 2.1 we showed an example where the continuous behaviour contains jumps. Since these jumps may occur at the composition points we present shortly a possibility to relax the condition and allow jumps at the composition point for the function describing the timewise behaviour.

For a zero-duration trajectory $\underline{x}$ we have $\underline{x} \cdot (d, g) = (d, g)$ if $x = g(0)$; otherwise the composition is undefined. Likewise, $(d, g) \cdot \underline{x} = (d, g)$ if $x = g(d)$ or $d = \infty$. That means that zero-duration trajectories are neutral elements with respect to composition if composition is defined.

For a discrete infinite set of durations D, e.g. $D = \mathbb{N} \cup \{\infty\}$, trajectories are isomorphic to nonempty finite or infinite words over the value set V (cf. Example 3.2.7). Moreover if V consists of values of computations, then the elements of the set of all processes can be viewed as sets of computation streams (e.g. [BS01]).

Definition 4.1.8 (process)
A *process* is a set of trajectories. The set of all processes is denoted by PRO. The greatest process, namely the set of all trajectories, is denoted by TRA.

We do not put any restrictions (such as prefix-closure) on a process. A process may be interpreted as a part or a whole hybrid system. The trajectories of the process then describe all possible behaviours of the system. Moreover, the trajectories of an arbitrary process can be classified according to their duration; either a trajectory has finite duration or its duration is infinite. The *purely finite* and *purely infinite parts* of a process A correspond to

$$\mathsf{fin}\, A =_{df} \{(d,g) \,|\, (d,g) \in A,\, d \neq \infty\} \quad \text{and} \quad \mathsf{inf}\, A =_{df} \{(d,g) \,|\, (d,g) \in A,\, d = \infty\}\ .$$

This definition entails, for an arbitrary process $A \in \mathsf{PRO}$, that $\mathsf{inf}\, A = A \cdot \emptyset$ and by complementation $\mathsf{fin}\, A = A - \mathsf{inf}\, A$. Hence a process is purely infinite, i.e., consists of infinite trajectories only, if $A = \mathsf{inf}\, A = A \cdot \emptyset$. Dually, a process B is purely finite, i.e., consists of finite trajectories only, if its purely infinite part is trivial, that is, if $\mathsf{inf}\, B = \emptyset$.

This characterisation allows us to lift the composition of trajectories to processes. This is done in a similar way as for omega languages (Example 3.2.7).

Definition 4.1.9 (composition of processes)
Composition is lifted to processes A, B as follows:

$$A \cdot B =_{df} \mathsf{inf}\, A \cup \{\tau_1 \cdot \tau_2, \,|\, \tau_1 \in \mathsf{fin}\, A,\, \tau_2 \in B,\, \tau_1 \cdot \tau_2 \text{ defined}\}\ .$$

Pointwise lifting is not possible, since this would imply $A \cdot \emptyset = \emptyset$ and hence the infinite part of A would disappear. Sets of zero-duration trajectories, corresponding to sets of values, can be used to restrict processes. Let R be such a set and A be an arbitrary process. Then $R \cdot A$ consists of those trajectories of A whose initial value lies in R, while $A \cdot R$ is the set of trajectories of A whose final value, if any, is in R. As we show in the next theorem those sets correspond to tests (cf. Section 3.5).

A restricted form of composition, the *chop*-operator $A^\frown B$, yields only trajectories that, after a finite trajectory of A, actually enter the second process. It is defined as $A^\frown B =_{df} (\mathsf{fin}\, A) \cdot B$, which implies $A \cdot B = (\mathsf{inf}\, A) \cup A^\frown B$.

Theorem 4.1.10

1. *The processes under union as addition and composition as multiplication form a Boolean weak quantale* $\mathsf{PRO} =_{df} (\mathcal{P}(\mathsf{TRA}), \cup, \cdot, \emptyset, I)$*, where I is the set of all zero-duration trajectories, i.e.,* $I = \{\underline{x} \,|\, x \in V\}$.

2. *Setting* $\mathsf{test}(\mathsf{PRO})$ *as* $\mathcal{P}(I)$, PRO *becomes a test weak quantale. In particular,* $\cdot$ *is universally disjunctive in its right argument.*

3. *Additionally, chop inherits the disjunctivity properties from* $\cdot$ *and is associative.*

The proof is by straightforward calculations. It can be found in Appendix B. Moreover, PRO forms a *full* quantale if the underlying time domain does not contain the special value ∞, i.e., only finite trajectories occur. Tests are sets of zero-duration trajectories describing sets of values. Moreover, we now see that the definition of purely (in)finite parts of processes fits perfectly to the general definition in the setting of lazy semirings and justifies the definitions of Section 3.4. Since PRO forms a quantale the largest purely infinite element and the largest purely finite element exist (Theorem 3.4.10). In PRO, $\mathsf{N} = \{(d,g) \mid d = \infty\}$ contains all elements with infinite duration and $\mathsf{F} = \{(d,g) \mid d < \infty\}$ contains all trajectories of finite duration. Lemma 3.2.9 implies that operators for finite and infinite iterations and all their laws are available for processes.

Corollary 4.1.11 PRO *also forms a weak Kleene and a weak omega algebra.*

In Lemma 3.4.15 we have presented various laws concerning purely finite and purely infinite parts. In particular, we showed that

$$\begin{aligned} \mathsf{inf}\, a^\omega &= (\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + \mathsf{inf}\, ((\mathsf{fin}\, a)^\omega) \;, \\ \mathsf{fin}\, a^\omega &= (\mathsf{fin}\, a)^\omega \sqcap \mathsf{F} \leq (\mathsf{fin}\, a)^\omega \;. \end{aligned}$$

Let us have a short look at the meaning of these laws in the case of processes. The former equation says that infinite behaviour results from entering an infinite part after a finite iteration of finite parts of the iterated process or by iterating finite parts of that process that all have long enough durations that their infinite iteration takes infinite duration. The second equation fits also well with intuition, since in PRO it means that Zeno effects (infinite iterations that take finite duration) can only occur when some trajectories in a process a are finite. In Section 4.4 we will look at Zeno effects in detail.

Example 4.1.12 To use trajectories for our introductory example of the temperature control (cf. Section 2.1), we set $V = \mathbb{R}$ and $D = \mathbb{R}_{\geq 0}$. and define two processes, one for each control mode:

$$\begin{aligned} A^{\mathrm{Off}} &=_{df} \{(d,x) \mid d \in D,\, \dot{x} = -0.1x\} \;, \\ A^{\mathrm{On}} &=_{df} \{(d,x) \mid d \in D,\, \dot{x} = 5 - 0.1x\} \;. \end{aligned}$$

A^{OFF} models all possible behaviours when the heater is off, whereas A^{ON} describes the thermostat when the heater is on. The (singleton) set of possible initial values is given by $R_{20} =_{df} \{\underline{20}\}$. Hence, we can formalise the starting sequence of the thermostat described above as

$$R_{20} \cdot A^{\text{OFF}} \cdot A^{\text{ON}} .$$

Note that so far we have not modelled jump and invariant conditions. For this we use tests and restrict the ranges of trajectories accordingly. Generally, we represent an interval of values as a set of zero-duration trajectories by setting

$$R_{[l,u]} =_{df} \{\underline{x} \mid x \in [l,u]\} .$$

Then the sequence "OFF–*jump*–ON" equals $A^{\text{OFF}} \cdot R_{[18,19]} \cdot A^{\text{ON}}$. This eliminates from the full composition $A^{\text{OFF}} \cdot A^{\text{ON}}$ all trajectories in which the temperature at the joining point is outside the interval $[18, 19]$.

Since we want to describe the whole behaviour of the thermostat, we use operators for finite and infinite iteration. Then the whole system is described by

$$R_{20} \cdot T^* \quad \text{or} \quad R_{20} \cdot T^\omega$$

where $T =_{df} A^{\text{OFF}} \cdot R_{[18,19]} \cdot A^{\text{ON}} \cdot R_{[21,22]}$. □

In such a way, any hybrid automaton can be replaced by a corresponding regular-like expression as long as the constraint $g_1(d_1) = g_2(0)$ holds. Unfortunately, this constraint for composing trajectories (d_1, g_1) and (d_2, g_2) is very restrictive in a number of situations. For example no jumps between control modes are allowed if one follows the above example.

As we have seen, even in most elementary hybrid systems jumps may occur at composition points. The extended temperature control described by Figure 2.3 sets the state of the switch when changing the control modes, hence a jump occurs. Therefore a method for modelling proper jumps is needed. To relax the composition of trajectories we introduce an additional compatibility relation.

Definition 4.1.13 (compatibility relation)
A *compatibility relation* $\asymp \subseteq V \times V$ describes the behaviour at the point of composition.

To model a more liberal form of composition that takes $\asymp$ into account, we extend a finite trajectory (d, g) at the right end, i.e., at time d, using the compatibility relation. To this end we express that, up to $\asymp$, we do not care about the exact final value $g(d)$. Therefore we inflate the original trajectory to a process that before time d agrees with the original trajectory, but shows all values admitted by $\asymp$ at time d.

Definition 4.1.14 ((right-)extended trajectory/process)
For a compatibility relation $\asymp$ and a finite trajectory (d, g) with $d < \infty$, the *(right-)extended trajectory* is a process defined as

$$(d,g)_{\asymp} =_{df} \{(d,\hat{g}) \,|\, \hat{g}(x) = g(x),\, x \in [0,d[\,,\, g(d) \asymp \hat{g}(d)\}\ .$$

Since for an infinite trajectory (d, g) a right composition partner does not matter anyway, we set $(d,g)_{\asymp} =_{df} \{(d,g)\}$ if $d = \infty$.

The composition of (d_1, g_1) and (d_2, g_2) considering the compatibility relation $\asymp$ is then the composition

$$(d_1,g_1)_{\asymp} \cdot \{(d_2,g_2)\}$$

over PRO which results in a process. This allows proper jumps at the connection point between two trajectories (d_1, g_1) and (d_2, g_2). This is meaningful, since jumps within trajectories are already allowed by our definition. We do not postulate any condition for $\asymp$. But in most cases $\asymp$ will be at least reflexive to accommodate the case of equal values $g_1(d_1)$ and $g_2(0)$. If one wants to *enforce* jumps at every composition point, $\asymp$ has to be irreflexive.

The operation is lifted pointwise to processes. From this we get $(A_{\asymp_1})_{\asymp_2} = A_{\asymp_1;\asymp_2}$, where ; is standard relation composition. Moreover the composition of two processes A and B considering a compatibility relation $\asymp$ is given by

$$A_{\asymp} \cdot B =_{df} \mathsf{inf}\, A \cup \bigcup_{a \in \mathsf{fin}\, A} a_{\asymp} \cdot B = \mathsf{inf}\, A \cup \mathsf{fin}\, A_{\asymp} \cdot B\ .$$

We have decided not to incorporate the compatibility relation into the definition of trajectory composition (Definition 4.1.9), since it would be technically cumbersome to do so. Note that in general $(A \cdot B)_{\asymp} \neq A \cdot (B_{\asymp})$. However as long as there is some "progress" in B, the equation holds.

Lemma 4.1.15 *If a process B does not contain zero-duration trajectories the compatibility relation can be shifted to the outermost level of multiplication, i.e., for arbitrary processes A and B*

$$B \cap I = \emptyset \ \Rightarrow\ A \cdot B_{\asymp} = (A \cdot B)_{\asymp}\ .$$

The proof is immediate by the definition of right-extended processes. Symmetrically, we could also employ the compatibility relation at the left end or even at both ends of a trajectory (process). But, the expressiveness cannot be improved dramatically.

Theorem 4.1.16 *Each left-extended process (trajectory) occurring after some duration $d > 0$ can be replaced by right-extended processes (trajectories).*

Proof. Every single trajectory is isomorphic to the process that only contains this trajectory. Hence we can restrict ourselves to processes. Moreover, it is sufficient to look at the product of two processes. The remaining cases immediately follow by distributivity and star/omega properties.

Following the above definition, a left-extended trajectory is given by

$$\succsim(d,g) =_{df} \{(d,\hat{g}) \,|\, \hat{g}(x) = g(x),\, x \in \mathsf{intv}\, d,\, x \neq 0,\, \hat{g}(0) \asymp g(0)\} \,.$$

This definition is again lifted pointwise to processes.

We now assume some process B with $B \cap I = \emptyset$ and determine $B \cdot \succsim A$. By splitting A into its test part $P =_{df} A \cap I$ and its test-free part $A' =_{df} A \cap \overline{I}$ and by distributivity we get

$$B \cdot \succsim A = B \cdot \succsim (P + A') = B \cdot (\succsim P + \succsim A') = B \cdot \succsim P + B \cdot \succsim A' \,.$$

By definition of left- and right-extended processes, the second summand equals $B_{\succsim} \cdot A'$. Since P only contains zero-duration trajectories, $\succsim P = P_{\succsim^{\mathsf{T}}}$, where $\asymp^{\mathsf{T}}$ denotes the converse of the relation $\asymp$. $P_{\succsim^{\mathsf{T}}}$ is itself a test. Hence

$$B \cdot \succsim A = B \cdot P_{\succsim^{\mathsf{T}}} + B_{\succsim} \cdot A' \,,$$

which uses only right extensions. □

The only proper improvement of left extentions is therefore at the very beginning of a trajectory. But this seems not useful at all, since the compatibility relation was design to perform proper jumps, which occur at the right ends of trajectories. By this theorem we also see that extending a trajectory at both ends does not improve the expressiveness.

Example 4.1.17 In the extended thermostat example we set $V = \mathbb{R} \times \{0,1\}$, where the first component of a pair represents the temperature and the second one the value of the switch. We define the compatibility relation as

$$\asymp =_{df} \{((x,0),(x,1)) \,|\, x \in \mathbb{R}\} \cup \{((x,1),(x,0)) \,|\, x \in \mathbb{R}\} \,.$$

The set of trajectories that start in the mode Off and then go to Down is described by

$$A^{\mathrm{Off}}_{\succsim} \cdot A^{\mathrm{Down}} \,,$$

where A^{Off} and A^{Down} describe the behaviours inside the modes Off and Down. In particular, $A^{\mathrm{Off}} =_{df} \{(d,g) \,|\, d \in D,\, g(t) = (x(t),1),\, \dot{x} = 0.1x\}$ and A^{Down} can be constructed in a similar way. □

In such a way, any arbitrary hybrid automaton can now be replaced by a corresponding regular-like expression. This is shown in the next section.

Independent of the development of the algebra of hybrid systems presented so far, the notion of *hybrid programs* was developed [Pla08]. Hybrid programs are an extension of discrete regular programs or dynamical logic ([HKT00]) by continuous evolutions. An overview of the syntax of hybrid programs is given in Figure 4.2.

Statement	Effect
$\alpha;\beta$	sequential composition, first performs α and then β afterwards [β never starts if α does not terminate]
$\alpha \cup \beta$	nondeterministic choice, following either α or β
α^*	nondeterministic repetition, repeating α $n \geq 0$ times
$x := \theta$	discrete assignment of the value of term θ to variable x (jump)
$x := *$	non-deterministic assignment of an arbitrary real number to x
$(x_1', \ldots, x_n', F)$	continuous evolution of x_i along differential equation system $x_i = \theta_i$ restricted to maximum domain or invariant region F
$?F$	check if formula F holds, do nothing otherwise
if(F) then α	perform α if F holds, do nothing otherwise
if(F) then α else β	perform α if F holds, perform β otherwise

Figure 4.2: Statements of hybrid programs [PQ08]

It is easy to see that there is the same correspondence between hybrid programs and our algebra as there is between Kleene algebra and discrete regular programs. The first three lines are clearly connected. Lines 4 and 5 can be modelled using the compatibility relation. The continuous evolution is encoded inside trajectories in our setting. It may be restricted at the beginning or at the ending by some tests or, if the evolution should be restricted to some region F, one can use the operator $\square$ which is presented below. The remaining three statements can be modelled using tests as done for the discrete case (cf. [Koz97]). For example, an ordinary if-statement "if p then a else b" becomes $p \cdot a + \neg p \cdot b$. Platzer also defines parametrised modal operators $[\alpha]$ and $\langle\alpha\rangle$ for hybrid programs [Pla08]. But these operators are just the counterparts for the modal operators defined for domain semirings in [DMS06] and for domain lazy semirings in [Möl07] (see also end of Section 3.6). Therefore the presented algebra here also forms the algebra for hybrid programs. However, our algebra allows us to distinguish between purely finite and infinite parts and offers an operator for infinite iterations. As mentioned before, this operator reflects a main feature of hybrid systems. The authors of [Pla08, PQ08] argue that each infinite loop can be approximated by a finite iteration (if the iteration is "long enough"). This assumption may be correct if one excludes Zeno behaviour; it is not correct if such behaviours occur. We will give an example in Section 4.4.

4.2 Embedding Hybrid Automata

Hybrid automata (see Section 2.2) can be modelled in the algebraic setting in a generic way using these concepts. Consider a hybrid automaton of dimension n with control graph (M, E). Then as value set we choose $V =_{df} M \times \mathbb{R}^n$.

In a given mode $v \in M$ the behaviour of the automaton in the interval $[0, d]$ coincides with a trajectory (d, g) such that $g(t) = (v, f(t))$ for some function $f : [0, d] \to \mathbb{R}^n$ that satisfies the invariant and flow conditions of v. This corresponds to Henzinger's relation $(v, f(0)) \xrightarrow{d} (v, f(d))$ defined in [Hen96].

The compatibility relation is given by

$$(v, x) \asymp (w, y) \Leftrightarrow_{df} (v = w \wedge x = y) \vee (\exists\, e \in E : ({}^\bullet e = v \wedge e^\bullet = w \wedge \mathit{jump}(e)(x, y))) ,$$

where again ${}^\bullet e$ and $e^\bullet$ denote the source and the target of an edge e. The first part $(v = w \wedge x = y)$ deals with compositions that do not leave a control mode and the second part models the event belonging to the edge e (if the edge is present). For readability, we introduce new predicates: $\mathit{stay}((v, x), (w, y)) \Leftrightarrow_{df} v = w \wedge x = y$ for the first and $\mathit{go}((v, x), e, (w, y)) \Leftrightarrow_{df} \exists\, e \in E : ({}^\bullet e = v \wedge e^\bullet = w) \wedge \mathit{jump}(e)(x, y)$ for the second part. The compatibility relation is then given by

$$k \asymp l \Leftrightarrow_{df} \mathit{stay}(k, l) \vee \mathit{go}(k, e, l) , \tag{4.1}$$

where $k = (v, x)$ and $l = (w, y)$ are elements from the value set $V = M \times \mathbb{R}^n$.

Basic construction. Without loss of generality we restrict the generic construction to hybrid automata with exact one initial mode, denoted by v_0. The construction of an algebraic expression from a given automaton now proceeds by the following steps:

- For each control mode v of the automaton we define a process

$$\begin{aligned} A^v =_{df} \{(d, g) \mid\ & d \in D, \forall\, t \in \mathsf{intv}\, d : g(t) = (v, f(t)), \\ & f : \mathsf{intv}\, d \to \mathbb{R}^n, \forall\, t \in \mathsf{intv}\, d : \mathit{flow}(v)(f(t), \dot{f}(t)), \\ & \forall\, x \in \mathsf{ran}\ f : \mathit{inv}(v)(x)\} . \end{aligned}$$

  Moreover, we define a zero-duration process for the initial condition:

$$A^{init} =_{df} \{(0, g) \mid\ g(0) = (v_0, f(0)),\ \mathit{init}(v_0)(f(0))\} .$$

- For each A^v determine $A^v_{\asymp}$ with $\asymp$ as above.
- Next we construct a finite-state machine. The set of vertices is $M \cup \{v_0', v_0''\}$, i.e., the finite-state machine has, next to the vertices of the hybrid automaton, two additional states. The transition system is given as follows:

– If there is an edge from u to v in the hybrid automaton, then the finite-state machine has an edge from u to v which is labelled with A^v.

– The initial condition of the hybrid automaton is transformed as follows:

where i denotes the initial condition of the hybrid automaton.

- Kleene has shown how to construct a regular expression from a given automaton [Kle56]. A formal and constructive algorithm, called *state elimination technique* is e.g. given in [HMU07, RS97]. Since we have constructed a finite-state machine in the last step we can directly use this algorithm.

  While in the original construction the star for finite iteration is used, hybrid automata does not distinguish between finite and infinite iteration. Hence one has to decide, whenever iteration occurs, whether it should be finite iteration (A^*), infinite iteration (A^ω) or the choice between both ($A^* + A^\omega$)[5].

- Note, that hybrid automata may include Zeno effects. Therefore such effects might also occur in the corresponding algebraic expressions. Having such a behaviour $^\omega$ is sometimes not adequate and has to be replaced by another operator. We discuss this in detail in Section 4.4.

Although the focus of this book is on the algebraic calculi for hybrid systems, we show the described transformation for a concrete example step by step. Later we only present the results and skip the page-filling and tedious calculations.

Example 4.2.1 We follow the above basic construction and derive an algebraic expression for the extended thermostat automaton (Figure 2.3).

In our example the time of durations is $\mathbb{R}^+$ and only two variables occur; hence $V = M \times \mathbb{R}^2$, where $M = \{\text{OFF}, \text{ON}, \text{DOWN}\}$. To improve readability we denote elements of V by brackets and reduce the size of the parentheses inside, e.g. $[\text{ON}, (x, y)]$. Otherwise far too many parentheses would occur.

- We first define three processes — one for each control mode. To improve readability we solve the differential equations and give the explicit definition of the

[5] The operation which offers the choice between finite and infinite iteration is similar to von Wright's omega operator [vW02]. The connection between both is discussed in [HMS06].

functions.

$$
\begin{aligned}
A^{\textsc{On}} &=_{df} \{(d,g) \mid\ d \in \mathbb{R}^+, \forall\, t \in \mathsf{intv}\, d : g(t) = [\textsc{On}, f(t)],\\
&\qquad f(t) = ((x_0 - 50)\cdot \exp(\tfrac{-t}{10}) + 50, y_0),\ x_0, y_0 \in \mathbb{R},\\
&\qquad \forall\,(x,y) \in \mathsf{ran}\ f : x \leq 22\}\ ,\\
A^{\textsc{Off}} &=_{df} \{(d,g) \mid\ d \in \mathbb{R}^+, \forall\, t \in \mathsf{intv}\, d : g(t) = [\textsc{Off}, f(t)],\\
&\qquad f(t) = (x_0 \cdot \exp(\tfrac{-t}{10}), y_0),\ x_0, y_0 \in \mathbb{R}\\
&\qquad \forall\,(x,y) \in \mathsf{ran}\ f : x \geq 18\}\ ,\\
A^{\textsc{Down}} &=_{df} \{(d,g) \mid\ d \in \mathbb{R}^+, \forall\, t \in \mathsf{intv}\, d : g(t) = [\textsc{Down}, f(t)],\\
&\qquad f(t) = (x_0 \cdot \exp(\tfrac{-t}{10}), y_0),\ x_0, y_0 \in \mathbb{R}\}\ .
\end{aligned}
$$

In the defined processes, the variables x_0 and y_0 allow arbitrary starting values for the functions f and g, respectively. That means that the processes consist of a whole bunch of functions. Further we have to define a zero-duration process that corresponds to the initial condition:

$$A^{init} =_{df} \{(0,g) \mid g(0) = [\textsc{Off}, (20,1)]\}\ .$$

- Next we define the compatibility relation $\asymp$:

$$
\begin{aligned}
\asymp\ =_{df} \{\ &([\textsc{On},(x,y)]), ([\textsc{On},(x,y)]),\ ([\textsc{Off},(x,y)], [\textsc{Off},(x,y)]),\\
&([\textsc{Down},(x,y)], [\textsc{Down},(x,y)]),\\
&([\textsc{On},(x,y)], [\textsc{Down},(x,0)]),\ ([\textsc{On},(x_2,y)], [\textsc{Off},(x_2,y)]),\\
&([\textsc{Off},(x,y)], [\textsc{Down},(x,0)]),\ ([\textsc{Off},(x_1,y)], [\textsc{On},(x_1,y)]),\\
&([\textsc{Down},(x_1,y)], [\textsc{On},(x_1,1)]),\ ([\textsc{Down},(x_2,y)], [\textsc{Off},(x_2,1)])\ \}\ ,
\end{aligned}
$$

where x, y are arbitrary elements of $\mathbb{R}$, $x_1 \in \mathbb{R}, x_1 < 19$ and $x_2 \in \mathbb{R}, x_2 > 21$. The first two lines deal with compositions that do not leave a control mode; the third line describes the out-going edges of On, the fourth the edges of Off and the last lines models the switches coming from Down. By this, $A^{\textsc{On}}_{\asymp}$, $A^{\textsc{Off}}_{\asymp}$ and $A^{\textsc{Down}}_{\asymp}$ can be determined easily.

- Following the third item of the basic construction we determine the finite-state machine depicted in Figure 4.3.

- Figure 4.4 illustrates the state elimination technique which yields the algebraic expression $A^{init} \cdot A^{\textsc{Off}} \cdot \left(A^{\textsc{Down}} \cdot A^{\textsc{Off}} + A^{\textsc{Loop}}\right)^{\omega}$, where $A^{\textsc{Loop}}$ is an abbreviation for $(A^{\textsc{On}} + A^{\textsc{Down}}\,;A^{\textsc{On}}) \cdot \left((A^{\textsc{Down}} \cdot A^{\textsc{On}})^* + (A^{\textsc{Down}} \cdot A^{\textsc{On}})^{\omega}\right) \cdot (A^{\textsc{Off}} + A^{\textsc{Down}} \cdot A^{\textsc{Off}})$. For the iteration resulting from the elimination of mode On (occurring in the term $A^{\textsc{Loop}}$), we allow finite and infinite iteration. For the outer iteration we force the system to have an infinite loop, since the whole system

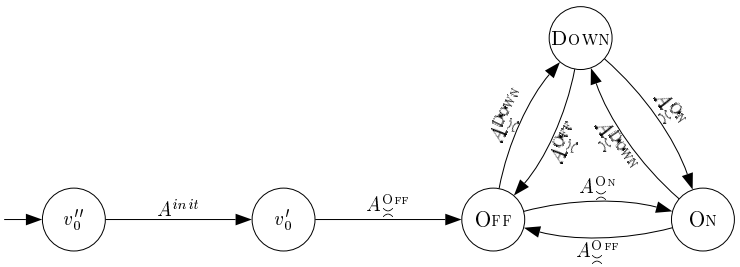


Figure 4.3: Constructed finite-state machine

should never end and iterate forever. By distributivity and Lemma 3.4.5(1) this is equivalent to

$$A^{init} \cdot \big(A^{\text{Off}}_{\gtrless} \cdot A^{\text{Down}}_{\gtrless} + (\widehat{A}^{\text{On}}_{\gtrless} + \widehat{A}^{\text{Down}}_{\gtrless} \cdot A^{\text{On}}_{\gtrless}) \cdot (A^{\text{Down}}_{\gtrless} \cdot A^{\text{On}}_{\gtrless})^* \cdot (1 + A^{\text{Down}}_{\gtrless}) (\widehat{A}^{\text{On}}_{\gtrless} + \widehat{A}^{\text{Down}}_{\gtrless} \cdot \widehat{A}^{\text{On}}_{\gtrless}) \cdot (\widehat{A}^{\text{Down}}_{\gtrless} \cdot \widehat{A}^{\text{On}}_{\gtrless})^{\omega} \cdot (1 + \widehat{A}^{\text{Down}}_{\gtrless})\big)^{\omega} . \qquad \square$$

Often, it is not necessary to store the control mode in the value set, i.e., V can be chosen as $\mathbb{R}^n$ instead of $M \times \mathbb{R}^n$ (e.g. Example 4.1.17).

4.3 Composing Hybrid Systems Algebraically

As we have seen, more complicated hybrid systems are built by combining smaller systems. Moreover, we have recapitulated the product and variants of parallel composition of hybrid automata (cf. Section 2.4). We now discuss the algebraic counterparts of such constructions.

The product construction was similar to the one on finite-state machines. Hence, at first glance, the algebraic counterpart seems to be the product semiring. For two (lazy) semirings $(A, +_A, 0_A, \cdot_A, 1_A)$ and $(B, +_B, 0_B, \cdot_B, 1_B)$ the *product (semiring)* is defined as

$$(A \times B, +_\times, (0_A, 0_B), \cdot_\times, (1_A, 1_B)) ,$$

where $+_\times$ and $\cdot_\times$ are componentwise operators. By standard results from universal algebra (e.g., [BS81, Part II §7]) the product structure indeed forms a (lazy) semiring.

Although this works fine for finite-state machines and regular expressions, it does not work for our case. The reason is that synchronisation can only occur at switches. This means that either both automata have to perform a transition or none. Since,

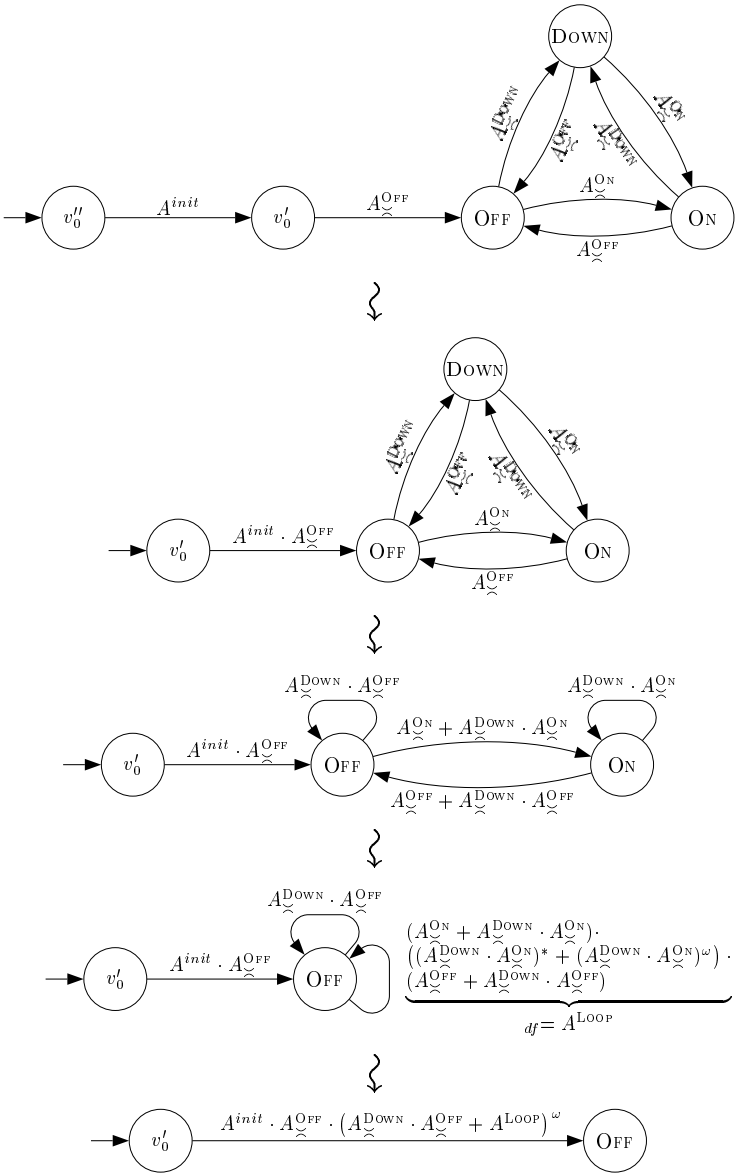


Figure 4.4: State elimination technique by an example

in contrast to finite state machines, hybrid automata can stay in a mode for a certain amount of time, synchronisation requires reasoning about durations. Both systems have to stay within their current modes for the same duration; then either both or none have to perform a transition.

For trajectories $\tau_1 = (d_1, g_1)$ and $\tau_2 = (d_2, g_2)$ with $d_1, d_2 \in D$ we define $\tau_1 \shuffle \tau_2$

$$\tau_1 \shuffle \tau_2 =_{df} \begin{cases} (d_1, g_1 \nabla g_2) \text{ if } d_1 = d_2 \\ \text{undefined} \quad \text{otherwise} , \end{cases}$$

where $(f \nabla g)(x) =_{df} (f(x), g(x))$ is just the parallel execution. As usual, this operation can be lifted pointwise to processes. We see that in the case of two semirings of processes with the same set of durations the parallel-composed trajectories form again trajectories. Viz., if the first process contains only trajectories τ_1 with functions $g_1 : \mathsf{intv}\, D \to V$ and for all trajectories τ_2 of the second process we have $g_2 : \mathsf{intv}\, D \to V'$, then the parallelised process semiring contains trajectories with functions of type $\mathsf{intv}\, D \to V \times V'$. The cross product avoids the problem of shared variables by duplicating them.

Using this we can now formalise algebraic versions for the composition operations presented in Section 2.4. Since all composition operations are based on the Cartesian product, they can be handled in a similar way. We use a slightly modified version of the basic construction.

As in the base case we define for all modes v of H_1 and H_2 processes P^v. The elements for the product semiring are then defined as

$$P^{v_1 v_2} =_{df} P^{v_1} \shuffle P^{v_2} ,$$

where v_1 and v_2 are modes of H_1 and H_2, respectively. As we have seen, transitions (and thus events) can be modelled by a compatibility relation $\asymp$. It can be defined similar to Equation (4.1) and is used to determine $P^{v_1 v_2}_{\asymp}$. Then we follow the remaining lines of the basic construction.

Depending of the kind of composition the compatibility relation is defined as follows.

A transition of the product of two hybrid automata can only be performed if both involved automata perform a transition. If one of them does not perform a transition the composed automaton is not allowed to do so. For two elements of the value set $V_1 \times V_2$ of the composed automaton (k_1, k_2) and (l_1, l_2) the compatibility relation is then given by

$$\begin{aligned} (k_1, k_2) \asymp (l_1, l_2) \Leftrightarrow_{df} & \\ & (stay(k_1, l_1) \wedge stay(k_2, l_2) \vee \\ & (go(k_1, e_1, l_1) \wedge go(k_2, e_2, l_2) . \end{aligned}$$

As in Equation (4.1), the first part deals with compositions that do not leave a control mode. The second part models that both automata involved perform a transition.

For the strict parallel composition we have also to consider the composition relation. This yields more cases for the compatibility relation.

$$\begin{aligned}(k_1,k_2) \asymp (l_1,l_2) \;\Leftrightarrow_{df}\;& \\ & (stay(k_1,l_1) \;\wedge\; stay(k_2,l_2)) \;\vee \\ & (go(k_1,e_1,l_1) \;\wedge\; go(k_2,e_2,l_2) \;\wedge\; e_1 \parallel e_2) \;\vee \\ & (go(k_1,e_1,l_1) \;\wedge\; stay(k_2,l_2) \wedge\; \nexists\, e_2 \in E_2 : e_1 \parallel e_2) \;\vee \\ & (stay(k_1,l_1) \;\wedge\; go(k_2,e_2,l_2) \wedge\; \nexists\, e_1 \in E_1 : e_1 \parallel e_2)\;.\end{aligned}$$

As before the first line handles compositions not leaving a control mode. The remaining lines follow the case distinction of Definition 2.4.3. The compatibility relation for the liberal parallel composition can be defined in a similar straightforward way. Hence all composition operations for hybrid automata can be embedded in the algebraic model.

Now we have a look at composing processes in general. First, we look at "unsynchronised" parallel runs of hybrid systems and generalise the above definition for parallel execution of trajectories. For trajectories $\tau_1 = (d_1, g_1)$ and $\tau_2 = (d_2, g_2)$ with $d_1, d_2 \in D$ we first define $\tau_1 \mathbin{|} \tau_2$ for some special cases:

$$\tau_1 \mathbin{|} \tau_2 \;=_{df}\; \begin{cases} (d_1, g_1 \nabla g_2) & \text{if } d_1 = d_2 \\ (d_2, \mathsf{const}_{d_2}(g_1) \nabla g_2) & \text{if } d_1 = 0 \\ (d_1, g_1 \nabla \mathsf{const}_{d_1}(g_2)) & \text{if } d_2 = 0\;, \end{cases}$$

where $\mathsf{const}_d(f)(x) = f(0)$ is the constant function on $[0, d]$. Often the above definition is sufficient. But, sometimes one also has to consider the cases $0 < d_1 < d_2$ or $0 < d_2 < d_1$. For those cases there are some choices and decisions to be made. For example, should the trajectories start at a common time point or should they end after the same duration?

If $d_1 < d_2$, then, by definition of the order on D, there exists $d_3 \in D$ with $d_1 + d_3 = d_3 + d_1 = d_2$. Therefore the trajectory $\tau_1 = (d_1, g_1)$ can be lengthened to duration d_2 using constant trajectories as

$$\tau_1 \cdot (d_3, g_3) \qquad \text{or} \qquad (d_3, g_3') \cdot \tau_1\;,$$

where $g_3(x) = g_1(d_1)$ and $g_3'(x) = g_1(0)$. Using the first of these products in the parallel composition $(\tau_1 \cdot (d_3, g_3)) \mathbin{|} \tau_2$ means that the trajectories τ_1 and τ_2 start at a common time point, whereas $((d_3, g_3') \cdot \tau_1) \mathbin{|} \tau_2$ enforces that τ_1 and τ_2 end together. Again this operation can be lifted to processes.

Next, we want to synchronise trajectories via reachable events, i.e., events that have to occur after a finite duration. In the general case we model this by a set X of shared variables. Post-multiplying with the process

$$\{(0, g_1 \triangledown g_2) \mid \ g_1|_X = g_2|_X\} \ ,$$

where $|_X$ restricts the domains to X, enforces synchronisation.

Synchronisation of infinite trajectories can only be done after a finite initial duration. In the case of hybrid automata the set of durations is $\mathbb{R}_{\geq 0}$. Hence each infinite trajectory τ_0 has prefixes of arbitrary duration, i.e., for all infinite trajectories τ_0 and all $d \in \mathbb{R}_{\geq 0}$ it holds that

$$\exists\, \tau_1, \tau_2 : \ \tau_0 = \tau_1 \cdot \tau_2 \ , \text{ and duration of } \tau_1 = d \ .$$

Therefore, one can use the synchronisation for finite trajectories also for infinite ones. Synchronisation after an infinite amount of time does not make sense.

An example for composing hybrid systems is given in Chapter 6, where we will revisit the railway examples of Chapter 2.

4.4 Zeno Effects

Zeno of Elea's famous paradox of Achilles and the tortoise is well known. However, with few exceptions (e.g. [JELS99, AAS05]) authors do not treat Zeno effects within hybrid systems in detail, even if they appear in their theoretical models. As we have seen, even quite simple systems, like a bouncing ball, contain Zeno effects (cf. Section 2.3). In [PQ08] the authors avoid Zeno effects for the bouncing ball by changing the setting and making the damping factor a variable which can change between each jump. Therefore the given hybrid program and the presented hybrid automaton are not equivalent in a strict formal way. In this section we present a possible way of handling Zeno effects in PRO and characterise the Zeno and Zeno-free parts of hybrid systems.

Roughly spoken, a Zeno effect occurs if an infinite iteration does not take infinite duration. To speak about such phenomena we can use the purely finite and purely infinite parts of processes. Furthermore, it is useful to determine A^ω for a process $A \in \mathsf{PRO}$.

Lemma 4.4.1 *In the setting of a lazy omega algebra, $a^\omega = a$ if a is purely infinite.*

For an arbitrary process infinite iteration can be determined by the general decomposition law $a^\omega = (\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + (\mathsf{fin}\, a)^\omega$ (see Lemma 3.4.15(3)). Therefore it suffices to determine A^ω for purely finite processes A.

To do so, we need to define prefixes of trajectories. For that we embed PRO into an algebra of guarded strings, where the behaviour of omega is well known. After determine omega in that model we use a projection to go back to the algebra of hybrid systems. In Section 3.2.7, we introduced the lazy semiring $\mathsf{GS}(P, \Sigma)$ of guarded strings over two alphabets P and Σ. We now have a look at $\mathsf{GS}(\mathsf{test}(\mathsf{PRO}), \mathsf{fin}\,(\mathsf{TRA}))$ over all *finite* trajectories. To define an embedding of purely finite processes into this lazy structure we define for a trajectory $\tau = (d, g)$ with finite duration d a function

$$\iota : (d, g) \;\mapsto\; \underline{g(0)} \,.\, (d, g) \,.\, \underline{g(d)}$$

that maps τ to a guarded string of length 3. Here $v.w$ denotes concatenation of v and w, as describes in the examples of Section 3. As usual, we lift ι pointwise to a function $\iota : \mathsf{fin}\,(\mathsf{TRA}) \to \mathsf{GS}(\mathsf{test}(\mathsf{PRO}), \mathsf{fin}\,(\mathsf{TRA}))$. The above definition preserves the composition condition, i.e., $\tau_1 \cdot \tau_2$ is defined if and only if $\iota(\tau_1) \bowtie \iota(\tau_2)$ is defined. Further by pointwise lifting we get the following result.

Corollary 4.4.2 *The mapping ι is disjunctive. In particular, $\iota(A+B) = \iota(A)+\iota(B)$.*

The composition of different images of purely finite processes then yields alternating sequences of $\mathsf{test}(\mathsf{TRA})$ and $\mathsf{fin}\,(\mathsf{TRA})$. The sequences might have infinite length. A zero-duration trajectory $\underline{x}$ is also mapped to a guarded string of length 3, namely $\underline{x} \,.\, \underline{x} \,.\, \underline{x}$. Hence $\iota(I) = \{\underline{x} \,.\, \underline{x} \,.\, \underline{x} \,|\, x \in V\}$.

Next we construct a homomorphism from finite guarded strings to processes. Later on we will extend this to infinite strings. For finite guarded strings a projection from $(\mathsf{test}(\mathsf{PRO}) \times \mathsf{TRA})^* \times P \to \mathsf{TRA}$ is inductively defined by

$$\phi(\underline{x}) = \underline{x} \qquad \text{and} \qquad \phi(w.\tau) = \phi(w) \cdot \tau \;,$$

where $x \in V$ and $\tau \in \mathsf{TRA}$. Here τ also covers the case that τ is a test. By this definition we immediately get $\phi(u \bowtie w) = \phi(u) \cdot \phi(w)$. Lifting ϕ pointwise yields the following result.

Lemma 4.4.3 *$\phi : \mathsf{FGS}(\mathsf{test}(\mathsf{PRO}), \mathsf{TRA}) \;\mapsto\; \mathsf{PRO}$ is a Kleene algebra homomorphism, i.e., $\phi(0) = 0$, $\phi(1) = 1$, $\phi(a + b) = \phi(a) + \phi(b)$, $\phi(a \cdot b) = \phi(a) \cdot \phi(b)$ and $\phi(a^*) = \phi(a)^*$. Moreover by pointwise lifting, ϕ is also disjunctive.*

Proof. Except the equation for finite iteration all calculations are straightforward and can be shown either by definition or follow from pointwise lifting. The last equation is by fixpoint fusion (cf. Proof of Theorem 3.2.9). We choose $f(x) = \phi(a) \cdot x + \phi(1)$, $g(x) = \phi(x)$ and $h(x) = a \cdot x + 1$. By definition all these functions are isotone and g is continuous. Moreover we have

$$g(h(x)) \;=\; g(a \cdot x + 1) \;=\; \phi(a \cdot x + 1) \;=\; \phi(a) \cdot \phi(x) + \phi(1) \;=\; f(\phi(x)) \;=\; f(g(x)) \;.$$

The third step follows by additivity and multiplicativity of ϕ. Hence by fixpoint fusion we have $g(\mu h) = \mu f$. In particular, we have for an element $a \in \mathsf{FGS}(\mathsf{test}(\mathsf{PRO}), \mathsf{TRA})$

$$\phi(a^*) = g(\mu h) = \mu f = \phi(a)^* \ .$$

□

Moreover, $\phi(\iota(A)) = A$ for an arbitrary process A. By universal algebra a Kleene algebra homomorphism preserves (in)equations.

Obviously ϕ cannot be extended to infinite guarded strings, since the inductive definition does not work. We define ϕ of an infinite guarded string by the supremum of its finite prefixes, i.e., we calculate the "limit" of all prefixes. A prefix of a guarded string is defined as usual: $w_1 \in (P \times \Sigma)^* \times P$ is a finite prefix of w_2 if and only if there is a $u \in (P \times \Sigma)^* \times P \cup (P \times \Sigma)^\omega$ such that $w_1 \bowtie u = w_2$. In signs $w_1 \sqsubseteq w_2$. Infinite guarded strings are maximal with respect to this order. Moreover, in GS each (infinite) guarded string w can be determined by the supremum of all its (finite) prefixes.

$$w \ = \ \sup\{u \,|\, u \sqsubseteq w\} \ = \ \sup\{u \,|\, u \sqsubseteq w, \, |u| < \infty\} \ . \tag{4.2}$$

If w has finite length the set of prefixes is finite, hence w is the maximum. The homomorphism ϕ is $\sqsubseteq$-isotone, i.e.,

$$w_1 \sqsubseteq w_2 \ \Rightarrow \ \phi(w_1) \sqsubseteq \phi(w_2) \ . \tag{4.3}$$

More generally, we define

Definition 4.4.4 (prefix relation)
The *prefix relation* $\sqsubseteq$ between trajectories $\tau_1 = (d_1, g_1)$ and $\tau_2 = (d_2, g_2)$ is defined as

$$\tau_1 \sqsubseteq \tau_2 \ \ \Leftrightarrow_{df} \ \ d_1 \leq d_2 \wedge g_2|_{\mathsf{intv}\, d_1} = g_1 \ ,$$

where the stroke $|_X$ means function restriction to subset X.

The first conjunct on the right hand side is equivalent to $\mathsf{intv}\, d_1 \subseteq \mathsf{intv}\, d_2$.

Lemma 4.4.5 *The prefix relation $\sqsubseteq$ is a partial order with $\tau_1 \sqsubseteq \tau_2$ if and only if $\exists\, \tau_3 : \tau_1 \cdot \tau_3 = \tau_2$. Moreover, if composition is right-isotone with respect to that order, i.e., if $\tau_1 \sqsubseteq \tau_2$ then $\tau_3 \cdot \tau_1 \sqsubseteq \tau_3 \cdot \tau_2$. Infinite trajectories are maximal with respect to this order.*

The proof is straightforward using the definition of the prefix relation. Let us go back to determine A^ω for a process A. To describe infinite concatenations of trajectories from a purely finite process A, we use the homomorphism ϕ and the fact that each guarded string is the limit of its prefixes. We build a set of all prefixes of A^ω, i.e., sets

of longer and longer trajectories that agree in their initial parts with some trajectory of A^ω.

$$\mathsf{PRE}(A) =_{df} \bigcup_{w \in (\iota(A))^\omega} \{\phi(u) \mid u \sqsubseteq w, |u| < \infty\} .$$

By Equation (4.3), we see that $\{\phi(u) \mid u \sqsubseteq w, |u| < \infty\}$ is indeed a set of prefixes of a single trajectory of A^ω. For convenience we denote such a set by $\mathsf{pre}(w)$, where w is a guarded string. Since each element of $(\iota(A))^\omega$ has infinite length, w is either empty or has infinite length (cf. Example 3.2.7). Moreover, there are an infinite number of prefixes, i.e., $|\mathsf{pre}(w)| = \infty$. Infinite iteration then results by passing to some sort of "limit" of of $\mathsf{pre}(w)$. Unfortunately, in contrast to Equation (4.2), the supremum of $\mathsf{pre}(w)$ need not exist in PRO. We illustrate this fact by the following example.

Example 4.4.6 Consider the process $A =_{df} \{(\frac{1}{n^2}, g) \mid g(x) = n^2 \cdot x + n,\ n \in \mathbb{N}\}$, where the time domain D and the value set V are equal to $\mathbb{R}_{\geq 0}$. By definition of the embedding $(\iota(A))^\omega$ only consists of one single element, namely

$$\underline{1} \cdot (1, g) \cdot \underline{2} \cdot (\frac{1}{4}, g) \cdot \underline{3} \cdot (\frac{1}{9}, g) \ldots .$$

All finite prefixes of this infinite guarded string have the form $u = \underline{1} \cdot (1, g) \ldots \underline{n}$ $(n \in \mathbb{N})$. By this $\phi(u)$ has duration $\sum_{i=1}^{n} \frac{1}{i^2}$. The supremum of these trajectories is a trajectory over a right-open interval of duration $d_T =_{df} \sum_{i=1}^{\infty} \frac{1}{i^2} = \frac{\pi^2}{6}$; hence the supremum does not exist in PRO. Completing the open interval to closed one yields the problem of defining $g(d_T)$. Shortly we will define an extended supremum that allows all possible values at d_T. □

Theorem 4.4.7 *Let A be purely finite process and let $H : \mathsf{PRO} \to \mathsf{PRO}$ be a function defined by $H(X) =_{df} A \cdot X$.*

1. *Let X be expanded by H, i.e., assume $X \subseteq H(X)$. Then for every $\xi \in X$ there is a guarded string $w \in \mathsf{GS}$ such that the set of prefixes $\mathsf{pre}(w) \subseteq \mathsf{PRE}(A)$ and $\tau \sqsubseteq \xi$ for all $\tau \in \mathsf{pre}(w)$.*
2. *$A^\omega = \{\xi \in \mathsf{TRA} \mid \exists\, w \in \inf \mathsf{GS} : \mathsf{pre}(w) \subseteq \mathsf{PRE}(A), \forall\, \tau \in \mathsf{pre}(w) :\ \tau \sqsubseteq \xi\}$.*

The proof cannot be automatised within a first-order setting. It can be found in Appendix B.

The fact that A^ω contains arbitrary extensions of infinite A-iterations also explains why the property $A^\omega = A^\omega \cdot \top$ (see Section 3.4) is not completely unnatural: for arbitrary $B \in \mathsf{PRO}$ the process $B \cdot \top$ is the extension closure of B. Hence $A^\omega = A^\omega \cdot \top$ reflects the fact that, operationally, after a Zeno gap the behaviour doesn't matter, since the gap cannot be "crossed" anyway.

Now, we generalise from PRO to a weak omega algebra S.

Definition 4.4.8 (Zeno-free/Zeno elements)
An element a of a weak omega algebra is called *divergent* or *Zeno-free*, if $a^\omega \leq \mathsf{N}$. It is called *Zeno* if it is not Zeno-free and it is called *convergent* if $a^\omega \leq \mathsf{F}$.

The least element 0 is the only element which is convergent, divergent and Zeno-free, since $0^\omega = 0$.

Lemma 4.4.9 *In full omega algebra (where* 0 *is also a right annihilator) every element is convergent.*

However, A^ω is not completely adequate for reasoning about and exclusion of Zeno effects. For many purposes its extension-closedness gets in the way, since it yields a too loose description of infinite iteration. For that reason we introduce another iteration operator † (in words: dagger) which narrows down the set of possible behaviours. However, in contrast to omega, its definition works up to now only for special time domains.

Next we define a supremum-operator for $\mathsf{pre}(w)$ which equals the proper supremum if possible, otherwise it completes the open interval to a closed one. This is done by a construction similar to the one used in Section 4.1 for the treatment of proper jumps. However the definition works only for special time domains. Let again A be purely finite and assume that the time domain D is complete, i.e., contains suprema for all its subsets. We set $d_\mathrm{T} =_{df} \sup\{d \,|\, (d,g) \in \mathsf{pre}(w)\}$.

Definition 4.4.10 (extended supremum)
For a set of trajectory-prefixes $\mathsf{pre}(w) = \{\phi(u) \,|\, u \sqsubseteq w, |u| < \infty\}$, we define the *extended supremum* $\widehat{\sup} : \mathsf{PRO} \to \mathsf{PRO}$ by

$$\widehat{\sup}(\mathsf{pre}(w)) =_{df} \begin{cases} \{\sup(\mathsf{pre}(w))\} & \text{if } d_T = \infty \\ \{(d_T, g)\} & \text{if } (d_T, g) \in \mathsf{pre}(w) \\ \{(d_T, \hat{g}) \,|\, \hat{g}(d_T) = v,\ v \in V, & \\ \qquad \exists\, (d,g) \in \mathsf{pre}(w) : & \text{otherwise .} \\ \qquad \hat{g}(t) = g(t) \text{ if } t \leq d_T\} & \end{cases}$$

If $d_\mathrm{T} = \infty$ the limit of the set of prefixes do not show a Zeno effect and the result is a singleton process consisting just of one infinite trajectory. For $d_\mathrm{T} \neq \infty$, two cases arise. The first case can only happen when the sequence of prefixes becomes stationary with infinitely many trajectories of duration zero and identical value v at the end. This means a special kind of Zeno behaviour, viz. "stepping on the spot" forever. The second case, where $d_T \neq \{d \,|\, (d,g) \in \mathsf{pre}(w)\}$, i.e., $d_\mathrm{T} > d$ for all trajectories $(d,g) \in \mathsf{pre}(w)$, means "proper" Zeno behaviour where the trajectories become longer and longer without ever reaching the "limit time" d_T.

ϕ can be extended to infinite guarded strings by setting

$$\phi(w) = \widehat{\sup}(\mathsf{pre}(w)) \text{ if } |w| = \infty \ ,$$

that can again be lifted pointwise to sets of guarded strings. Unfortunately, ϕ is not a homomorphism any longer, since in general $\phi(u \bowtie w) \neq \phi(u) \cdot \phi(w)$ if u has infinite length. However, ϕ is still multiplicative and $\phi(u \bowtie w) = \phi(u) \cdot \phi(w)$ if u is finite (w might be infinite).

Corollary 4.4.11 *If A is a purely finite process then $A^\omega = \phi((\iota(A))^\omega) \cdot \top$.*

Definition 4.4.12 (infinite iteration for processes w.r.t. Zeno behaviour) For a purely finite process A, we define $A^\dagger =_{df} \phi((\iota(A))^\omega)$. $A^\dagger$ of an arbitrary process is defined $A^\dagger$ via $A^\dagger = (\mathsf{fin}\, A)^* \cdot \mathsf{inf}\, A + (\mathsf{fin}\, A)^\dagger$ (cf. Lemma 3.4.15(3)).

This gives another characterisation for infinite iteration in PRO, which respects Zeno behaviour. With this construct, Zeno effects can be excluded by considering only the properly infinite trajectories in $\mathsf{inf}\, A^\dagger = A^\dagger \cap \mathsf{N}$. This could not be achieved reasonably with A^ω, since that includes trajectories which are infinite because they add an arbitrary infinite behaviour to a Zeno initial part. This is made precise by Part (1) of Theorem 4.4.7. Since the definition is based on omega iteration on GS and projection ϕ we get for an arbitrary set of guarded strings $L \in \mathsf{GS}(\mathsf{test}(\mathsf{PRO}), \mathsf{fin}\,(\mathsf{TRA}))$

$$\phi(L^\omega) \ = \ (\phi(L))^\dagger \tag{4.4}$$

if $L \cap \mathsf{test}(\mathsf{PRO}) = \emptyset$. Moreover, from Definition 4.4.12 we get immediately

Corollary 4.4.13 *Infinite iteration of a zero-duration process is stationary, that is $P^\dagger = P$. In particular we have $I^\dagger = I$ for the multiplicative identity I of* PRO*, whereas $I^\omega = \top = \mathsf{TRA}$.*

As we have seen, Zeno effects may occur in hybrid automata and therefore such effects might also occur in the corresponding algebraic expressions. To avoid such behaviour one can replace omega by dagger in the base construction of Section 4.2 and apply a meet operation with the set of all infinite trajectories at the outermost level as we will do in Example 4.4.17.

Theorem 4.4.14 *Let H be as in Theorem 4.4.7.*

1. *$A^\dagger$ is a fixpoint of H.*
2. *Let X be expanded by H, i.e., assume $X \subseteq H(X)$. Then every $\tau \in X$ has a prefix in $A^\dagger$.*
3. *$A^\omega = A^\dagger \cdot \top$.*

Again, the proof can be found in the Appendix. Up to now it is open if $A^\dagger$ is a special fixpoint. We know that it is in general neither the least nor the greatest solution of $a \cdot x = x$. We can image that this fixpoint is an optimal fixpoint in the sense of Manna and Shamir [MS75, MS76]. However, we could neither verify or falsify this conjecture so far nor give a purely algebraic (first-order) characterisation.

An immediate consequence of Part (3) is that $A^\dagger$ and A^ω coincide if A is Zeno-free.

Lemma 4.4.15 *For an arbitrary process A*

$$A^\dagger \leq \mathsf{N} \Leftrightarrow A^\omega \leq \mathsf{N} \Rightarrow A^\dagger = A^\omega \ .$$

The following properties for dagger are consequences of omega properties (Lemmas 3.4.3 and Lemma 3.4.5) and definition of $\dagger$, ι and ϕ.

Lemma 4.4.16 *Assume arbitrary processes A and B and a zero-duration process P.*

1. $A \subseteq B \Rightarrow A^\dagger \subseteq B^\dagger$.
2. $A^* \cdot A^\dagger = A^\dagger$.
3. $A \cdot (B \cdot A)^\dagger = (A \cdot B)^\dagger$.
4. $(A \cdot B)^\dagger \leq (A + B)^\dagger$.
5. $(A^+)^\dagger = A^\dagger$.
6. $(A + B)^\dagger = (A^* \cdot B)^\dagger + (A^* \cdot B)^* \cdot A^\dagger$.

Proof. We only give the proof of the last item. The others are similar. Moreover, we assume again A and B to be purely finite. For arbitrary processes the claim follows again from Lemma 3.4.15(3). To increase readability and to reduce brackets we write ιA and ιB instead of $\iota(A)$ and $\iota(B)$.

The claim follows by straightforward calculations using the properties derived so far, e.g. Corollary 4.4.2 and Lemma 4.4.3.

$$\begin{aligned}
(A + B)^\dagger &= \phi\big(\big(\iota(A + B)\big)^\omega\big) \\
&= \phi\big((\iota A + \iota B)^\omega\big) \\
&= \phi\big(((\iota A)^* \cdot \iota B)^\omega + ((\iota A)^* \cdot \iota B)^* \cdot (\iota A)^\omega\big) \\
&= \phi\big(((\iota A)^* \cdot \iota B)^\omega\big) + \phi\big(((\iota A)^* \cdot \iota B)^* \cdot (\iota A)^\omega\big) \\
&= \big(\phi((\iota A)^* \cdot \iota B)\big)^\dagger + \phi\big(((\iota A)^* \cdot \iota B)^* \cdot (\iota A)^\omega\big) \\
&= (A^* \cdot B)^\dagger + (A^* \cdot B)^* \cdot \phi\big((\iota A)^\omega\big) \\
&= (A^* \cdot B)^\dagger + (A^* \cdot B)^* \cdot A^\dagger
\end{aligned}$$

□

Example 4.4.17 For the algebraic version of the temperature control (cf. Example 4.1.12), we can now describe all non-Zeno behaviours as $R_{20} \cdot T^{\dagger} \sqcap \mathbb{N}$, where T equals again $A^{\mathrm{Off}} \cdot R_{[18,19]} \cdot A^{\mathrm{On}} \cdot R_{[21,22]}$. □

With the next example we want to point out the specific differences between the three iteration operators. For this, we return to the bouncing ball example of Section 2.3.

Example 4.4.18 Following the lines of the above construction, we set $D = \mathbb{R}_{\geq 0}$ and $V = \mathbb{R}^2$ and define for the control mode a process

$$A^{\mathrm{FLY}} =_{df} \{(d, (x_1, x_2)) \mid \dot{x}_1 = x_2,\ \dot{x}_2 = -g\} .$$

The whole automaton can then be modelled by

$$A \cdot (A^{\mathrm{FLY}}_{\asymp})^{\dagger} .$$

The compatibility relation is defined by $\asymp =_{df} \{((0, x), (0, -0.9x)) \mid x \in \mathbb{R}\}$ and the initialisation A by $\{\underline{(h, 0)}\}$. The system $A \cdot (A^{\mathrm{FLY}}_{\asymp})^{\dagger}$ is a singleton process containing only one trajectory. It is sketched by Figure 2.4. By standard techniques from real analysis, the convergence point (the point where the ball does not bounce back) is at time point $C =_{df} \frac{1+c}{(1-c)g}\sqrt{2hg}$.

Using the other kinds of iterations for the automaton yields obviously different behaviours. The system $A \cdot (A^{\mathrm{FLY}}_{\asymp})^{\omega}$ with omega as iteration operator contains an infinite number of trajectories. First they coincide with the trajectory of $A \cdot (A^{\mathrm{FLY}}_{\asymp})^{\dagger}$, but starting from time point C some miraculous behaviour occurs. This means that the ball might lie on the ground forever or somebody can lift the ball to a new initial altitude or something else may happen. Finally, the star operator $(A \cdot (A^{\mathrm{FLY}}_{\asymp})^{*})$ yields only prefixes of complete bounces. Hence it only contains trajectories of duration less than (not equal to) C.

We now change the setting and model the following situation. After time C the ball is picked up and dropped again from the initial altitude. Obviously, this time point is reached even if an infinite number of bounces occurs. A corresponding hybrid automaton is given in Figure 4.5. It uses a clock x_3 to measure when time C is reached.

The algebraic counterpart avoids introducing a clock and equals

$$\left(A \cdot (A^{\mathrm{FLY}}_{\asymp})^{\dagger}_{\asymp'}\right)^{\dagger} ,$$

where $\asymp' =_{df} \{((0, x), (h, 0)) \mid x \in \mathbb{R}\}$. As we will show in the next section this system reaches the value $(h, 0)$ infinitely often and therefore it is alive. This property cannot be shown if one uses finite iteration. □

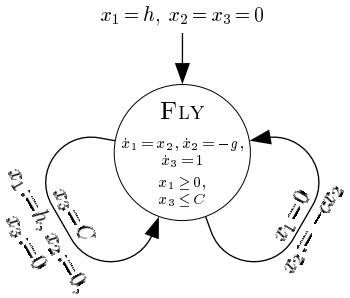


Figure 4.5: Modified bouncing ball

4.5 Safety and Liveness

In the previous section we restricted processes to their Zeno-free parts. Now, we want to deal with the general case that a process is restricted by an additional condition. Abstractly, let A stand for the process and C for the condition; then we want to form the meet $A \sqcap C$. If A is a composite process we want to distribute the condition to its components if possible. If A is a sum this is easy. However, if A is a product, we need special conditions for C to do this.

Definition 4.5.1 ((sub)modularity)
An element c is called *submodular* if $\forall\, a, b \in S \ :\ c \sqcap (a \cdot b) \leq (c \sqcap a) \cdot (c \sqcap b)$ and *modular* if the formula can be strengthend to an equation for arbitrary a and b.

We obtain useful characterisations of these properties. Since in the Boolean case $\mathsf{fin}\, a = a \sqcap \mathsf{F}$, Lemma 3.4.14(4) states that F is modular if the underlying semiring is weak.

The following lemma summarises elementary properties for submodular elements. They will be used in the remainder to prove useful statements concerning processes.

Lemma 4.5.2 *Assume a Boolean weak semiring (Kleene algebra).*

1. *The following properties are equivalent.*
   (a) Element $c \in S$ is submodular.
   (b) $(c \sqcap \mathsf{F}) \cdot \overline{c} + \overline{c} \cdot \top \leq \overline{c}$.
   (c) $\mathsf{F} \cdot \overline{c} \cdot \top \leq \overline{c}$.

   In particular, 1 *is submodular iff* $\overline{1} \cdot \overline{1} \leq \overline{1}$.

2. *Element $c \in S$ is modular iff it is submodular and* transitive, *i.e., satisfies $c \cdot c \leq c$. In particular, 1 is modular iff $\overline{1} \cdot \overline{1} \leq \overline{1}$.*
3. *Submodular elements are closed under addition and meet, hence form a lattice.*
4. *If c is modular then for all a we have $c \sqcap a^+ = (c \sqcap a)^+$ and $c \sqcap a^* = (c \sqcap a)^+ + (c \sqcap 1)$, with $b^+ =_{df} b \cdot b^*$.*
5. *If c is modular then $d \leq c$ is submodular iff $(c \sqcap \mathsf{F}) \cdot (c \sqcap \overline{d}) \cdot c \leq c \sqcap \overline{d}$.*

Since some parts of the above lemma are not easy to automate (one needs sophisticated additional hypotheses), we give the proof in the appendix.

By the shunting rule (Lemma 3.2.3) the property $\overline{1} \cdot \overline{1} \leq \overline{1}$ is equivalent to $1 \leq \overline{\overline{1} \cdot \overline{1}}$. The element $\overline{\overline{1} \cdot \overline{1}}$ has been called step in von Karger's work [vK98]; it represents the elements that cannot be decomposed into non-subidentities. Since we can think of the identity element 1 as a process that does not proceed in time, this property says that progress in time cannot be undone by composition.

Definition 4.5.3 (progressive (semiring))
A Boolean lazy semiring satisfying the property $\overline{1} \cdot \overline{1} \leq \overline{1}$ is called *progressive* or *progressive semiring*.

Often, time requirements are useful. For example, an event is guaranteed to happen after a certain time if one restrict the duration of the involved processes. One way of asserting this, is already given by the chop operator. Every trajectory in $A^\frown B$ guarantees that, unless Zeno effects occur, a suffix in process B is actually reached. To guarantee that B is reached after a certain time d one has to restrict A in a different way.

Example 4.5.4 Returning to the extended temperature control, we now want to guarantee that the heater is inactive for at most 30 time steps. Therefore we have to restrict A^{Down} by the process $A =_{df} \{(d, g) \,|\, d \leq 30, (d, g) \in \mathsf{TRA}\}$, i.e., we have to calculate $A^{\text{Down}} \sqcap A$. This process is the same as

$$\{(d, g) \,|\, d \leq 30, (d, g) \in A^{\text{Down}}\} \ .$$

Note that A is not submodular. □

Hence the Boolean structure gives a straightforward way to model time assertions using the meet operation.

Next to that, it may also be necessary to restrict the range of a process A. Here, the *range* $\mathsf{ran}\ A$ is defined as $\mathsf{ran}\ A =_{df} \bigcup_{t \in A} \mathsf{ran}\ t$. A general definition for range is given below.

Example 4.5.5 We now want to ensure that the heating controlled by the temperature control never leave the range $[18, 22]$.

We do this by observing that every subset W of the value set V is isomorphic to the process $P_W =_{df} \{\underline{x} \mid x \in W\}$. With $\top = \mathsf{TRA}$ and $\mathsf{F} = \mathsf{fin}(\mathsf{TRA})$ we define

$$\Diamond P_W =_{df} \mathsf{F} \cdot P_W \cdot \top \,, \qquad \Box P_W =_{df} \overline{\Diamond \neg P_W} \,.$$

Hence, $\Diamond P_W$ is the set of all trajectories that at some (finite) point in their time interval have a value in W, while $\Box P_W$ describes a safety aspect, viz. the set of all trajectories whose range satisfies the "invariant" W, i.e., $\Box P_W = \{\tau \mid \tau \in \mathsf{TRA}, \mathsf{ran}\ \tau \subseteq W\}$. Thus, the requested safety condition for the thermostat can be modelled as $\Box R_{[18,22]}$. Dually, $\Diamond P_W$ can be used to describe certain liveness aspects.

Looking again at the safety requirement of the thermostat we see that by the condition $A^{\text{OFF}} \cdot A^{\text{ON}} \leq \Box R_{[18,22]}$ we indeed restrict the range of $A^{\text{OFF}} \cdot A^{\text{ON}}$ as claimed. Using the meet

$$A^{\text{OFF}} \cdot A^{\text{ON}} \sqcap \Box R_{[18,22]}$$

is another way to enforce the restriction. □

For an algebraic characterisation of processes like P_W we use the idea of tests as introduced in Section 3.5. As shown in Lemma 4.1.10(2) $\mathsf{test}(\mathsf{PRO}) = \mathcal{P}(\{\underline{x} \mid x \in V\})$.

Lemma 4.5.6

1. *For $P \in \mathsf{test}(\mathsf{PRO})$ we have $P^\dagger = P$ and consequently $P^\omega = P \cdot \top$.*
2. *In PRO, the meet with a test distributes over composition, i.e., all tests in PRO are modular:*

$$P \in \mathsf{test}(\mathsf{PRO}) \;\Rightarrow\; P \cap A \cdot B = (P \cap A) \cdot (P \cap B) \,.$$

Proof. The first claim follows directly from Corollary 4.4.13 and Theorem 4.4.14. By isotony and transitivity of tests ($p \cdot p \leq p$), we get the general inequality

$$(p \sqcap a) \cdot (p \sqcap b) \leq p \cdot p \sqcap a \cdot b \leq p \sqcap a \cdot b \,.$$

The converse direction follows since tests in PRO correspond to zero-duration trajectories and 0 is indivisible. □

We have already used the tests of PRO for modelling restrictions and jump conditions in Section 4.1. Finally, it turns out that, even for arbitrary semirings, Part (2) is equivalent to the progressness condition introduced before:

Lemma 4.5.7 *All tests of a Boolean weak semiring S are modular iff S is progressive.*

Proof. ($\Rightarrow$) follows by Lemma 4.5.2(1), since 1 is a test.
($\Leftarrow$) Given test $p \leq 1$, by Lemma 4.5.2(2) the elements $c = 1$ and $d = p$ satisfy the assumptions in Lemma 4.5.2(5). Moreover, all tests are transitive. □

Using the concept of tests we now generalise the operators $\diamondsuit$ and $\Box$ to an arbitrary Boolean lazy semiring S. By Theorem 3.4.10 the greatest element $\top$, the greatest purely finite element F and the greatest purely infinite element N exist.

Definition 4.5.8 (diamond, box)
In a Boolean lazy semiring S we define for a test $p \in \mathsf{test}(S)$,

$$\diamondsuit p =_{df} \mathsf{F} \cdot p \cdot \top, \qquad \Box p =_{df} \overline{\diamondsuit \neg p} .$$

Further we introduce shorthands for the purely finite and purely infinite parts of boxes and set $\mathbb{F} p =_{df} \mathsf{fin}(\Box p) = \mathsf{F} \sqcap \Box p$, $\mathbb{N} p =_{df} \mathsf{inf}(\Box p) = \mathsf{N} \sqcap \Box p$ if S is separated.

Thus, $\Box p$ corresponds to the "always p" operator of von Karger [vK98], whence the notation. Since $\diamondsuit$ and $\Box$ do not yield tests as their results, they cannot be nested. This does no harm, since nested safety requirements do not seem to be useful anyway. All other algebraic operations, like addition and multiplication, are available for box and diamond. These operators are useful for hybrid systems (cf. Example 4.5.5). Hence our goal is to derive a number of useful algebraic laws for these operators. Later on we will give more examples. By definition it follows immediately that $\diamondsuit 0 = 0 = \Box 0$ and $\diamondsuit 1 = \top = \Box 1$. Another immediate consequence of the definitions is

Lemma 4.5.9 *For Boolean lazy semiring S and $p \in \mathsf{test}(S)$ the element $\Box p$ is submodular.*

Proof. By the definition of box we have $\mathsf{F} \cdot \overline{\Box p} \cdot \top = \mathsf{F} \cdot \mathsf{F} \cdot \neg p \cdot \top \cdot \top = \mathsf{F} \cdot \neg p \cdot \top = \overline{\Box p}$ and the claim follows from Lemma 4.5.2(1). □

The box operator shows useful and natural behaviour in the case of progressivity.

Lemma 4.5.10 *Let $p, q \in \mathsf{test}(S)$ in a progressive Boolean weak semiring S.*

1. $p \leq \Box q \Leftrightarrow p \leq q$.
2. $p \leq \Box p$.

By Lemma 4.5.6(2) and Theorem 4.1.10(1) PRO is progressive and Properties (1) and (2) hold. In $\mathsf{REL}(M)$, however, subidentities can be decomposed into non-subidentities (unless the underlying base set is a singleton); so these properties do not hold there.

Lemma 4.5.11 *Assume a Boolean lazy semiring S and $p \in \mathsf{test}(S)$. $\Box p = p \cdot (\Box p)$. If S is weak then also $\boxed{\mathsf{F}} p = p \cdot \boxed{\mathsf{F}} p$ as well as $\Box p = (\Box p) \cdot p$ and $\boxed{\mathsf{F}} p = \boxed{\mathsf{F}} p \cdot p$.*

Some of the following properties are satisfied only in a special kind of lazy semirings.

Definition 4.5.12 (safety-closed semiring)
A Boolean lazy semiring (quantale) S is called *safety-closed* if $(\Box p) \cdot (\Box p) \leq \Box p$.

Since elements of the form $\Box p$ correspond to safety properties, the notion is justified. In a safety-closed lazy Kleene algebra $\Box p$ is transitive and hence coincides with its own transitive closure, i.e., $(\Box p)^+ = \Box p$ (cf. Lemma 3.4.1(1)). Hence

$$a \leq \Box p \Leftrightarrow a^+ \leq \Box p \,.$$

Safety-closedness implies, next to other useful properties, that a composition satisfies a liveness assertion if that is satisfied in its first component or in the second component after some finite run of the first component.

Lemma 4.5.13 *Assume a Boolean weak semiring S that is safety-closed.*

1. *All boxes are modular.*
2. *All boxes are multiplicatively idempotent, i.e., $(\Box p) \cdot (\Box p) = \Box p$.*
3. *$\Box p \sqcap a^+ = (\Box p \sqcap a)^+$ and $\Box p \sqcap a^* = (\Box p \sqcap a)^+ + (\Box p \sqcap 1)$.*
4. *$\diamondsuit p \sqcap a \cdot b = (\diamondsuit p \sqcap a) \cdot b + a^{\frown}(\diamondsuit p \sqcap b)$.*

If S is also progressive, the term $\Box p \sqcap 1$ of Part (3) simplifies to p. The last line describes the situation where a product has to satisfy p somewhere. This can be the case if p is satisfied by the first component or by the second. In the latter case the second component has to be reached. Therefore the first component has to be finite, that means chop can be used. The dual of Part (4), namely that a composition satisfies a safety assertion if and only if its two components satisfy it ($\Box p \sqcap a \cdot b = (\Box p \sqcap a) \cdot (\Box p \sqcap b)$) follows immediately since boxes are modular (Part (1)).

Example 4.5.14 The safety requirement $R_{20} \cdot (A^{\text{OFF}} \cdot A^{\text{ON}})^+ \sqcap \Box R_{[18,22]}$ of Example 4.5.5 can be transformed into $R_{20} \cdot ((A^{\text{OFF}} \sqcap \Box R_{[18,22]}) \cdot (A^{\text{ON}} \sqcap \Box R_{[18,22]}))^+$ if PRO is safety closed. Hence, it suffices to guarantee the safety requirement for the two component processes A^{OFF} and A^{ON}. □

To prove that PRO satisfies $(\Box p) \cdot (\Box p) \leq \Box p$ we derive a sufficient criterion for safety-closedness For the technical developments of this we need additional operators.

Definition 4.5.15 (left residual)
In any lazy quantale, the *left residual* a/b exists and is characterised by the Galois connection

$$c \leq a/b \Leftrightarrow_{df} c \cdot b \leq a \ .$$

In PRO, this operation is characterised pointwise by $\tau_1 \in V/U \Leftrightarrow \forall \sigma \in U : \tau_1 \cdot \tau_2 \in V$ (provided $\tau_1 \cdot \tau_2$ is defined). Based on the right residual, in a Boolean lazy quantale a detachment operator can be defined.

Definition 4.5.16 (right detachment)
In a Boolean lazy semiring, the *right detachment* $a \lfloor b$ can be defined as

$$a \lfloor b =_{df} \overline{\overline{a}/b} \ .$$

The pointwise characterisation in PRO reads $\tau_1 \in V \lfloor U \Leftrightarrow \exists \tau_2 \in U : \tau_1 \cdot \tau_2 \in V$. Informally, this means that $V \lfloor U$ consists of trajectories which result from detaching a U-trajectory at the right from some V-trajectory. By de Morgan's laws, the Galois connection for $/$ transforms into the following exchange law for $\lfloor$.

Lemma 4.5.17 (detachment exchange) *In Boolean lazy semiring the exchange rule*

$$a \lfloor b \leq c \Leftrightarrow \overline{c} \cdot b \leq \overline{a}$$

holds. In particular, $(\Box p) \lfloor a \leq \Box p$ *and* $(\mathbb{F} p) \lfloor a \leq \mathbb{F} p$.

Lemma 4.5.17 generalises the Schröder rule of relational calculus. Intuitively, the two last claims mean that in PRO any prefix of a trajectory that satisfies a safety assertion again satisfies the assertion. Moreover, $\lfloor$ is isotone in both arguments and satisfies $a \lfloor 1 = a$.

Definition 4.5.18 (local linearity [vK98])
A Boolean lazy quantale is said to be *locally linear* if it satisfies

$$(a \cdot b) \lfloor c \ = \ a \cdot (b \lfloor c) + a \lfloor (c \lfloor b) \ .$$

The law describes the case analysis that appears when c is cut off $a \cdot b$ from the right. We distinguish two cases — c is a postfix of b or b is a postfix of c. We illustrate this behaviour in Figure 4.6, where the elements τ_1, τ_2, τ_3 are trajectories (singleton processes) of which only the time intervals are shown.

Local linearity of PRO can be proved as in the case of the semiring of formal languages, as done in [Höf03]. Hence, by the following lemma, PRO is safety-closed.

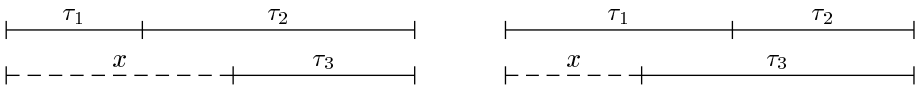


Figure 4.6: Local linearity

Lemma 4.5.19 *If S is a Boolean weak and locally linear quantale then S is safety-closed.*

Proof. By Lemma 4.5.11, isotony and Corollary 3.5.4, we get

$$\top \cdot \Box p \;=\; \top \cdot \Box p \cdot p \leq \top \cdot p \;=\; \overline{\mathsf{F} \cdot \neg p}\;.$$

Hence by the exchange rule (Lemma 4.5.17) $(\mathsf{F} \cdot \neg p) \lfloor \Box p \leq 0$. By the exchange rule again, definition, local linearity and isotony we get

$$\begin{aligned}
\Box p \cdot \Box p \leq \Box p \;&\Leftrightarrow\; (\diamondsuit \neg p) \lfloor \Box p \leq \diamondsuit \neg p \\
&\Leftrightarrow\; (\mathsf{F} \cdot \neg p \cdot \top) \lfloor \Box p \leq \diamondsuit \neg p \\
&\Leftrightarrow\; (\mathsf{F} \cdot \neg p) \cdot (\top \lfloor \Box p) + (\mathsf{F} \cdot \neg p) \lfloor ((\Box p) \lfloor \top) \leq \diamondsuit \neg p \\
&\Leftarrow\; \mathsf{F} \cdot \neg p \cdot \top + (\mathsf{F} \cdot \neg p) \lfloor \Box p \leq \diamondsuit \neg p \\
&\Leftrightarrow\; \diamondsuit \neg p + 0 \leq \diamondsuit \neg p \\
&\Leftrightarrow\; \mathsf{true}\;.
\end{aligned}$$

□

Sometimes one has safety properties of the form that first a predicate p has to be satisfied and afterwards another predicate q has to hold. The following laws are useful for checking whether a composition of processes satisfies such a condition.

Theorem 4.5.20 *Assume a Boolean weak and locally linear quantale S. Then for all $a, b \in S$ and $p, q \in \mathsf{test}(S)$ the following properties hold.*

1. $a \cdot b \sqcap \mathbb{F} p \cdot \Box q \;=\; (a \sqcap \mathbb{F} p) \cdot (b \sqcap \Box q) \;+\; (a \sqcap \mathbb{F} p) \cdot (b \sqcap \mathbb{F} p \cdot \Box q) \;+\; (a \sqcap \mathbb{F} p \cdot \Box q) \cdot (b \sqcap \Box q)$.
2. $a \cdot b \sqcap \mathbb{N} p \;=\; (a \sqcap \mathbb{N} p) + (a \sqcap \mathbb{F} p) \cdot (b \sqcap \mathbb{N} p) \;=\; (a \sqcap \Box p) \cdot (b \sqcap \mathbb{N} p)$.
3. $a \cdot b \sqcap \Box p \cdot \Box q \;=\; (a \sqcap \Box p) \cdot (b \sqcap \Box q) \;+\; (a \sqcap \Box p) \cdot (b \sqcap \Box p \cdot \Box q) \;+\; (a \sqcap \Box p \cdot \Box q) \cdot (b \sqcap \Box q)$.
4. *If additionally $p \leq \Box p$ holds, the summand $(a \sqcap \mathbb{F} p) \cdot (b \sqcap \Box q)$ can be omitted from the right hand sides of Parts (1) and (3).*

The lengthy proof as well as further properties of $\Box p$ can be found in Appendix B. The proof is not really hard, but it is too long for Prover9. For single, finite trajectories Part (1) is illustrated in Figure 4.7. Here, the change between properties p and q can occur either exactly at the composition point of a and b, inside a or inside b. That is why the formula on the right hand side of Part (1) consists of three summands.

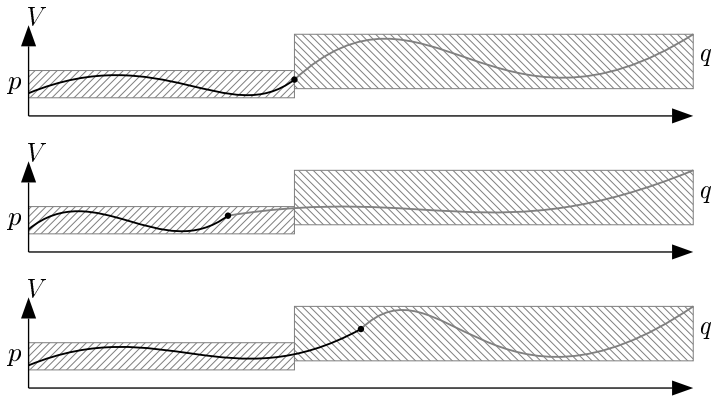


Figure 4.7: Composed trajectories satisfying $\mathbb{F}p \cdot \Box q$

An application of Lemma 4.5.20(1) is to combine safety requirements of the shape $R_{[l,u]}$. Since $\mathbb{F}p \cdot \Box q = \Box p^\frown \Box q$, a safety requirement of this form guarantees that the process $\Box q$ is actually entered.

The Box operator is not only useful for characterising safety properties it can also be used to define a general range operator for Boolean lazy semirings as introduced for PRO before. As a preparation we state the following.

Lemma 4.5.21 *Assume a lazy quantale in which $\cdot$ is also positively right-distributive. Then $\Diamond$ is universally disjunctive and $\Box$ is universally conjunctive. In particular, both operators are isotone.*

Proof. The property for $\Box$ follows by de Morgan's laws from the one for $\Diamond$, so we only show that. For nonempty set $L \subseteq P$ we get

$$\Diamond(\bigsqcup L) = \mathsf{F} \cdot (\bigsqcup L) \cdot \top = \bigsqcup(\mathsf{F} \cdot L) \cdot \top = \bigsqcup(\mathsf{F} \cdot L \cdot \top)$$

by positive right-disjunctivity and left-disjunctivity of $\cdot$. Moreover, we have

$$\Diamond(\bigsqcup \emptyset) = \mathsf{F} \cdot 0 \cdot \top = \mathsf{F} \cdot 0 = 0 = \bigsqcup \Diamond\emptyset$$

by left-strictness of $\cdot$ and $\mathsf{F} \cdot 0 = 0$. □

Definition 4.5.22 (range)
We can define a general operator $\mathsf{ran} : S \to \mathsf{test}(S)$ by the Galois connection

$$\mathsf{ran}\ a \leq p \Leftrightarrow_{df} a \leq \Box p\ .$$

Galois connections provide a number of useful properties for free (e.g., [Bir67, DP02, Bac02]). In particular, ran is universally disjunctive. Moreover we get immediately

Corollary 4.5.23 *The following properties hold:*

$$a \leq \Box(\mathsf{ran}\ a)\ , \qquad \mathsf{ran}\ (\Box p) \leq p\ , \qquad p \leq \Box p \Leftrightarrow \mathsf{ran}\ p \leq p\ .$$

Lemma 4.5.24 *If S is progressive then* $\mathsf{ran}\ p = p$.

Proof. By the third property of Corollary 4.5.23 it remains to show $p \leq \mathsf{ran}\ p$. Using the Galois connection of Definition 4.5.22 and Lemma 4.5.10(1), for arbitrary test q, we have

$$\mathsf{ran}\ p \leq q \Leftrightarrow p \leq \Box q \Leftrightarrow p \leq q\ .$$

Now setting $q = \mathsf{ran}\ p$ yields the claim. □

So far we have used Box and Diamond operators to model safety conditions. The introduced Diamond paves also the way to model and verify liveness conditions.

Standard liveness assumptions are that a given set of values (a zero-duration process) is reached an infinite number of times. Hence one would expect that formulas like $(\Diamond p)^\dagger$ or $(\Diamond p)^\omega$ occur. In the sequel we show that omega is again not adequate and that dagger should be used when Zeno effects might occur.

Lemma 4.5.25 *If S is a Boolean weak semiring and $p \in \mathsf{test}(S)$ then*

$$\Diamond p \cdot \Diamond p = \Diamond p\ .$$

This implies $(\Diamond p)^+ = (\Diamond p)^\omega = \Diamond p$.

Proof. ($\leq$) follows from isotony: $\Diamond p \cdot \Diamond p = \mathsf{F} \cdot p \cdot \top \cdot \Diamond p \leq \mathsf{F} \cdot p \cdot \top = \Diamond p$. The converse direction ($\geq$) by density of tests, neutrality of 1 and isotony:

$$\Diamond p = \mathsf{F} \cdot p \cdot \top \leq \mathsf{F} \cdot p \cdot p \cdot \top \leq \mathsf{F} \cdot p \cdot 1 \cdot 1 \cdot p \cdot \top \leq \mathsf{F} \cdot p \cdot \top \cdot \mathsf{F} \cdot p \cdot \top = \Diamond p \cdot \Diamond p\ .$$

The remaining claims then follow from Lemma 3.4.2 and Lemma 3.4.3(4). □

In contrast to intuition, formulas like $a \leq (\Diamond p)^\omega$ do not check wether a satisfies p infinitely often. It checks if p is reached once. The reason is that omega allows an

arbitrary trajectory after an infinite iteration. Hence the iteration can be seen as an infinite loop at p followed by some arbitrary trajectory.

In PRO, $(\diamondsuit P)^\dagger$ behaves different and shows again that the new iteration operator is more useful than omega. By unfold, it es straightforward that $(\diamondsuit P)^\dagger \leq \diamondsuit P$. More precisely we can determine $(\diamondsuit P)^\dagger$ by the following lemma.

Lemma 4.5.26 *In* PRO $(\diamondsuit P)^\dagger$ *can be split into three parts:*

$$\begin{aligned}(\diamondsuit P)^\dagger &= \mathsf{F}\cdot P\cdot \mathsf{N} + (\mathsf{F}\cdot P)^\dagger \\ &= \mathsf{F}\cdot P \;+\; \mathsf{F}\cdot P\cdot \mathsf{N} \;+\; ((\mathsf{F}\sqcap\overline{1})\cdot P)^\dagger\ .\end{aligned}$$

Proof. We first note that $(\mathsf{F}\cdot P)$ and $(\mathsf{F}\sqcap\overline{1})\cdot P$ are transitive, hence, by Lemma 3.4.2 $(\mathsf{F}\cdot P)^+ = (\mathsf{F}\cdot P)$ and $((\mathsf{F}\sqcap\overline{1})\cdot P)^+ = (\mathsf{F}\sqcap\overline{1})\cdot P$. Next, we have by Lemma 4.4.16(6), Lemma 4.4.1, unfold, distributivity, left annihilation of N and idempotence

$$\begin{aligned}(A+\mathsf{N})^\dagger &= (A^*\cdot\mathsf{N})^\dagger + (A^*\cdot\mathsf{N})^*\cdot A^\dagger \\ &= A^*\cdot\mathsf{N} + (1+(A^*\cdot\mathsf{N})\cdot(A^*\cdot\mathsf{N})^*)\cdot A^\dagger \\ &= A^*\cdot\mathsf{N} + A^\dagger + A^*\cdot\mathsf{N} \\ &= A^*\cdot\mathsf{N} + A^\dagger\ ,\end{aligned}$$

where A is an arbitrary process. By Lemma 4.4.16(3), splitting the greatest element and distributivity we can now show the first claim:

$$\begin{aligned}(\diamondsuit P)^\dagger &= (\mathsf{F}\cdot P\cdot\top)^\dagger = \mathsf{F}\cdot P\cdot(\top\cdot\mathsf{F}\cdot P)^\dagger \\ &= \mathsf{F}\cdot P\cdot(\mathsf{F}\cdot P+\mathsf{N})^\dagger \\ &= \mathsf{F}\cdot P\cdot(\mathsf{F}\cdot P)^*\cdot\mathsf{N} + (\mathsf{F}\cdot P)\cdot(\mathsf{F}\cdot P)^\dagger \\ &= \mathsf{F}\cdot P\cdot\mathsf{N} + (\mathsf{F}\cdot P)^\dagger\ .\end{aligned}$$

By modularity of 1 we have $\mathsf{F}\cdot P\sqcap 1 = P$. Moreover, by Lemma 3.5.3, $\mathsf{F}\cdot P\sqcap\overline{1} = (\mathsf{F}\sqcap\overline{1})\cdot P$. Hence we can split $\mathsf{F}\cdot P$ into $P+(\mathsf{F}\sqcap\overline{1})\cdot P$. Using Lemma 4.4.16(6) again, we get

$$\begin{aligned}(\mathsf{F}\cdot P)^\dagger &= (P+(\mathsf{F}\sqcap\overline{1})\cdot P)^\dagger \\ &= (P^*\cdot(\mathsf{F}\sqcap\overline{1})\cdot P)^*\cdot P^\dagger + (P^*\cdot(\mathsf{F}\sqcap\overline{1})\cdot P)^\dagger \\ &= ((\mathsf{F}\sqcap\overline{1})\cdot P)^*\cdot P + ((\mathsf{F}\sqcap\overline{1})\cdot P)^\dagger \\ &= (1+((\mathsf{F}\sqcap\overline{1})\cdot P)^+)\cdot P + ((\mathsf{F}\sqcap\overline{1})\cdot P)^\dagger \\ &= P+(\mathsf{F}\sqcap\overline{1})\cdot P + ((\mathsf{F}\sqcap\overline{1})\cdot P)^\dagger \\ &= \mathsf{F}\cdot P + ((\mathsf{F}\sqcap\overline{1})\cdot P)^\dagger\ .\end{aligned}$$

The third step follows from Lemma 3.5.5 ($p^* = 1$) and Corollary 4.4.13; the remaining steps are by unfold, distributivity and the above results. □

The first part ($\mathsf{F} \cdot P$) equals $\mathsf{F} \cdot P^\dagger$. Hence it models the situation of "stepping on the spot P" infinitely often. It may happen that before one reaches the ending point P is visited a couple of times. The second part describes the situation where, after P is reached at least once, one iteration step has infinite duration. The last part is the most interesting bit. It says that P is indeed reached an infinite number of time (with a non-zero-duration trajectory between each P). In this case "proper" Zeno-effects may occur, i.e. the trajectories of $\mathsf{F} \cdot P \cdot ((\mathsf{F} \sqcap \overline{1}) \cdot P)^\dagger$ can have finite and infinite duration. Typical trajectories for each part are sketched in Figure 4.8.

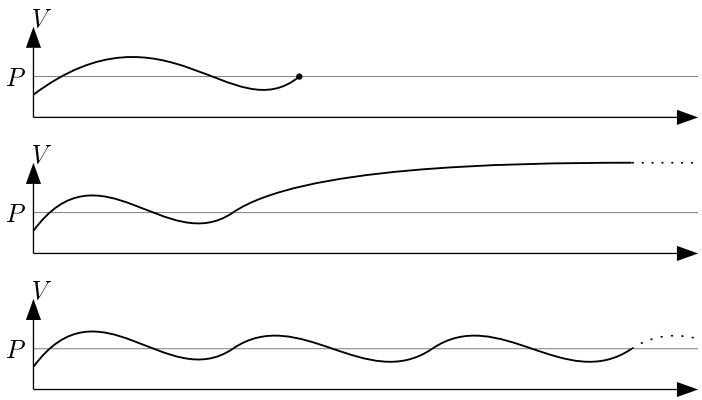


Figure 4.8: Different types of trajectories of $(\diamondsuit P)^\dagger$

By Lemma 4.5.26 we see that arguing about infinite occurrences of P is only possible if between each occurrence of P a finite, non-zero duration occurs. In other words we need a notion of progress within the diamond. This motivates to restrict $\diamondsuit P$ in a way that the iteration of this new construct only yields the third component of the previous lemma.

Definition 4.5.27 (progressive diamond)
For a Boolean lazy semiring S we define

$$\diamondsuit\!\!\!\!\!\!\diamondsuit p \;=_{df}\; \mathsf{fin}\,(\diamondsuit p) \sqcap \overline{1}\;.$$

If S is weak, we get by Lemma 3.4.14(4) $\diamondsuit\!\!\!\!\!\!\diamondsuit p = (\mathsf{F} \cdot p \cdot \mathsf{F}) \sqcap \overline{1}$. If S is even progressive then $\diamondsuit\!\!\!\!\!\!\diamondsuit p = (\mathsf{F} \cdot p \cdot \mathsf{F}) - p$. In PRO $(\diamondsuit\!\!\!\!\!\!\diamondsuit P)^\dagger$ and $(\diamondsuit\!\!\!\!\!\!\diamondsuit P)^\omega$ do again not coincide if Zeno

effects occur. To conclude this section we show that the above definition is indeed useful to verify liveness conditions.

Example 4.5.28 We return to the example of the iterated bouncing ball (cf. Example 4.4.18). The verification task is to show that the ball reaches infinitely often the initial position ($x_1 = h$ and $x_2 = 0$). This implies that the ball is dropped an infinite number of times. In PRO this property is given by

$$\big(\{\underline{(h,0)}\} \cdot (A^{\mathrm{FLY}}_{\succ})^{\dagger}_{\succ'}\big)^{\dagger} \leq (\diamondsuit\{\underline{(h,0)}\})^{\dagger} \ ,$$

where $\succ$ and $\succ'$ are as in Example 4.4.18. By isotony of composition and since $(A^{\mathrm{FLY}}_{\succ})^{\dagger}$ does not contain zero-duration trajectories, we get

$$\{\underline{(h,0)}\} \cdot (A^{\mathrm{FLY}}_{\succ})^{\dagger}_{\succ'} \leq \{\underline{(h,0)}\} \cdot \overline{I} \leq \diamondsuit\{\underline{(h,0)}\} \sqcap \overline{I} \ = \ \diamondsuit\{\underline{(h,0)}\} \ .$$

Then the claims follows by isotony of dagger (Lemma 4.4.16(1)). Therefore the system satisfies the given liveness property. In this particular example we know that $(\{\underline{(h,0)}\} \cdot (A^{\mathrm{FLY}}_{\succ})^{\dagger}_{\succ'})^{\dagger} \leq \mathsf{N}$; hence by Lemma 4.4.15 omega and dagger coincide. □

4.6 Specifications

"A specification is an explicit set of requirements to be satisfied" [Pyz03]. In case of hybrid systems, specifications are particular processes that express desired patterns. Algebraically, a trajectory τ satisfies a given specification W if and only if $\tau \in W$. Simple specifications are already given in the previous section by $\diamondsuit P$ and $\Box P$. To select those trajectories of a given process B which satisfy the specification W one can again use the meet operator $B \sqcap W$ to select all trajectories of a given process B that satisfies a specification W.

Example 4.6.1 We revisit the gear shift control of Section 2.3. When starting and accelerating a car it is the best to shift from the first to the second gear at around $3000\,rpm$; the following shifts should be around $4000\,rpm$. When constructing an automatic gear shift including a gear box, the specification has to guarantee that the rpm-range is not too low (the car would stutter) or to high (the engine might be damaged).

A typical specification W is sketched in Figure 4.9. Since we only want to give the idea we skip the details (as in some other examples). Visually, a trajectory satisfying the given specification has to lie inside the hatched area. □

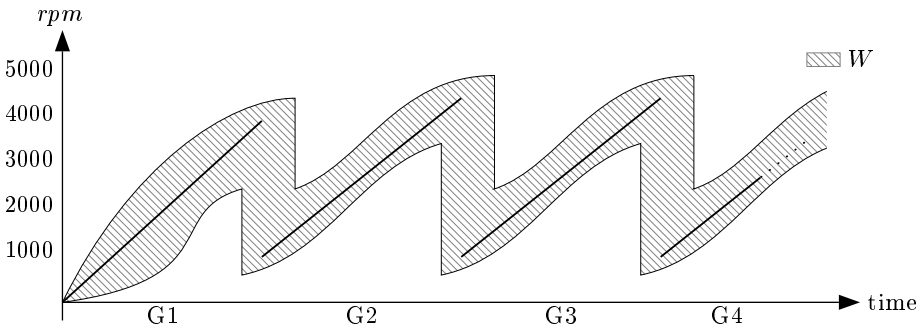


Figure 4.9: Specification of a gearbox

Following Sintzoff [Sin04], we define quantifier-like operators relating a specification W to a purported implementing process B. If one considers the values in V as states then the set

$$\{\underline{g(0)} \mid (d, g) \in B \sqcap W\} \tag{4.5}$$

gives all starting values of the trajectories in B admitted by W as well. This set is a test in the lazy semiring PRO.

To model such sets we use the concept of the abstract domain operator defined in Section 3.6. Informally, in PRO, the domain operator assigns to a set of trajectories the test that describes precisely its initial values.

The domain operation is guaranteed to exist in lazy quantales (cf. [DMS06]) and hence the following Corollary follows from Theorem 4.1.10(1).

Lemma 4.6.2 *Setting* $\ulcorner A = \{\underline{g(0)} \mid (d, g) \in A\}$*, the structure* PRO *forms a Boolean weak quantale with domain.*

Lemma 4.6.3 *If the underlying Boolean semiring* S *satisfies* $p \leq \Box p$ *(e.g.* S *is progressive) then* $\ulcorner(\Box p) = p$*.*

Proof. Axiom (d2) and Lemma 4.5.11 imply $\ulcorner(\Box p) \leq p$. The reverse inequation follows from the assumption $p \leq \Box p$, isotony of domain and $\ulcorner p = p$. □

Using the domain operation, Equation (4.5) compacts into $\ulcorner(B \sqcap W)$. Therefore, a first algebraic definition of Sintzoff's quantifiers reads as follows (the primes indicate

that we will use a different definition later on):

$$\mathsf{E}'B\,.\,W \;=_{df}\; {}^{\ulcorner}(B \sqcap W)\;, \tag{4.6}$$

$$\mathsf{A}'B\,.\,W \;=_{df}\; \neg\mathsf{E}'B\,.\,\overline{W} \;=\; \neg{}^{\ulcorner}(B \sqcap \overline{W})\;, \tag{4.7}$$

$$\text{Æ}'B\,.\,W \;=_{df}\; \mathsf{A}'B\,.\,W \;\sqcap\; \mathsf{E}'B\,.\,W\,. \tag{4.8}$$

This definition works in general Boolean domain lazy semirings. However, as the resulting quantifiers are operators of type $\mathsf{PRO} \to (\mathsf{PRO} \to \mathsf{test}(\mathsf{PRO}))$, they cannot easily be composed. Therefore, Sintzoff gives a different semantics to combinations of these quantifiers. We want to avoid this by introducing new quantifiers that omit the final projection into $\mathsf{test}(\mathsf{PRO})$. Doing this, we also allow a look into the "future" of trajectories and not only at the starting states. In other words, our new quantifiers in PRO should model formulas like

$$\tau_1 \in \mathsf{E}B\,.\,W \;\Leftrightarrow_{df}\; \exists\,\tau_2 \in B : \tau_1 \cdot \tau_2 \in W\;, \tag{4.9}$$

$$\tau_1 \in \mathsf{A}B\,.\,W \;\Leftrightarrow_{df}\; \forall\,\tau_2 \in B : \tau_1 \cdot \tau_2 \in W\;. \tag{4.10}$$

Hence, the process $\mathsf{E}B\,.\,W$ consists of all trajectories that can be completed by a B-trajectory to yield a trajectory in W. This fits perfectly with the concept of specifications described at the beginning of this section. Thus, $\mathsf{E}B\,.\,W$ is the inverse image of W under the operation $\cdot B$, while $\mathsf{A}B\,.\,W$ is the largest process whose image under $\cdot B$ is contained in W.

These quantifiers are operators of type $\mathsf{PRO} \to \mathsf{PRO}$ and their sequential composition simply is function composition. If, as with E' and A', a projection into $\mathsf{test}(\mathsf{PRO})$ is desired it can be added at the outermost level by finally applying the domain operator or one of the three quantifiers above. For their algebraic characterisation we basically want to use Equations (4.6) and (4.7), but express them with the help of detachment. Therefore we establish a connection between that and the domain operator.

Lemma 4.6.4 *In a Boolean lazy quantale, one has*

$${}^{\ulcorner}(b \sqcap w) \;=\; w\lfloor b \sqcap 1 \;=\; b\lfloor w \sqcap 1\;.$$

In the detachment formulas of this lemma, forming the meet with 1 performs the projection into the test algebra, and we obtain our revised operators by omitting this meet. There is a choice in which of these two formulas to use. We take the first one, since it results in a more direct translation of the universal quantifier A'.

Definition 4.6.5 (existential/universal continuations)
Assume a Boolean quantale S and $a, b \in S$. Then

$$\mathsf{E}b\,.\,w \;=_{df}\; w\lfloor b\;, \qquad \mathsf{A}b\,.\,w \;=_{df}\; \overline{\mathsf{E}b\,.\,\overline{w}} \;=\; w/b\;,$$
$$\text{Æ}b\,.\,w \;=_{df}\; (\mathsf{A}b\,.\,w) \sqcap (\mathsf{E}b\,.\,w)\;.$$

These quantifiers allow the following modal view: E is a kind of diamond, whereas A is a box operator. Correspondingly, we have the following properties that are typical of modal operators.

Lemma 4.6.6

1. $\mathsf{E}a.w$ *is universally disjunctive and* $\mathsf{A}a.w$ *is universally conjunctive in* w.
2. $\mathsf{E}(a \cdot b) \,.\, c = \mathsf{E}a \,.\, (\mathsf{E}b \,.\, c)$ *and* $\mathsf{A}(a \cdot b) \,.\, c = \mathsf{A}a \,.\, (\mathsf{A}b \,.\, c)$.
3. *If* $\cdot$ *is positively disjunctive in its right argument then* $\mathsf{E}a$ *is positively disjunctive and* $\mathsf{A}a$ *is positively antidisjunctive in* a.

In [Sin04], Sintzoff has used these operators to determine strategies in discrete-decision games. He has also shown that the theory of games can model reactive and hybrid systems. The interaction between the continuous and discrete dynamics corresponds to different moves of the game.

In detail, a control system can be presented as a game where the *controlling* and the *controlled* components are, respectively, the proponent and the opponent [Isa65]. As the controller has to counteract all possible failures induced by "moves" of the controlled system, it has to force the opponent into a "losing" position where nothing can go wrong anymore. In PRO, moves correspond to process transformers of the shapes $\mathsf{E}B$ and $\mathsf{A}B$. They describe the possible and guaranteed reachabilities from a game position using B-trajectories.

In the remaining section we briefly give some ideas and concepts of games. A more thorough analysis of the game-theoretic connection will be the subject of further research (cf. Section 7.2).

Abstractly, a *game* consists of one or more *players* who interact with each other. A *move* is an action of one player. Obviously, there are various kinds of games, like games with finite or infinite duration. In the second case, one can distinguish games with finite and infinite move duration. Another possibility of classifying games are the categories of *cooperate*, *non-cooperate* and *semi-cooperate* games, depending on the methods by which the players will interact. Further, we can split all games into *disjoint* and *non-disjoint* ones. Non-disjoint games allow several moves at the same time, while in a disjoint game there is one move at a time. We restrict ourselves to disjoint games with finite move duration.

In a *game round*, each player, one by one, makes a move. Hence, if S_i is defined as the a move of player i, a game round is represented by $(S_1 \cdot S_2 \cdots S_n)$. In that case, we can use the star, omega and dagger; $(S_1 \cdot S_2 \cdot \cdots \cdot S_n)^*$ describes a finite game and $(S_1 \cdot S_2 \cdot \cdots \cdot S_n)^\omega$ and $(S_1 \cdot S_2 \cdot \cdots \cdot S_n)^\dagger$ games with infinitely many game rounds. In the latter cases, the game has infinite duration if the S_i have positive durations and no Zeno-effects occur.

In a game with player X and opponent Y, represented by their respective moves $\mathsf{E}a$ and $\mathsf{A}b$, we can interpret a game round in which X has the possibility of "winning" as the product $\mathsf{E}a \circ \mathsf{A}b$ (cf. [DMS04a]), where $\circ$ is composition of process transformers. Finite or infinite games can then be described as $(\mathsf{E}a \circ \mathsf{A}b)^*$ or $(\mathsf{E}a \circ \mathsf{A}b)^\omega$ from which winning and losing "positions" can be calculated by fixpoint iteration (e.g according to *Kleene's theorem*); for details see e.g. [BM04, DMS06]. Since we have now sketched the connection to the modal view of games started in [BM04] and treated abstractly in [DMS04a], we could re-use the analysis of winning and losing positions provided in these papers. This will enable us to unify several results (e.g. [Sin04]).

Chapter 5

Logics for Hybrid Systems

空穴来风未必无因
(Chinese proverb)

This chapter starts with a short overview how different logics are used to model, specify and verify hybrid systems. Prominent tools in the analysis of concurrent, reactive and hybrid systems are *computation tree logic* and *neighbourhood logic*. Although they are by now well-understood, one rarely finds algebraic treatments of their semantics. In particular, no uniform handling is available.

We provide compact closed semantic expressions for computation tree logic by using modal operators in combination with (in)finite iteration. It is based on lazy quantales. Hence it fits perfectly in the setting of the algebra of hybrid systems presented before.

After that, we provide compact closed semantic expressions for neighbourhood logic. This is again done on the basis of lazy quantales. The laziness of the algebraisation extends the logic by infinite intervals and infinite behaviour. Originally neighbourhood logic was based on *finite* intervals and was not able to express all properties for infinite systems. The presented embedding immediately provides embeddings of all chop-based interval temporal logics like *interval logic* and *interval temporal logic*.

We close this chapter by applying the introduced theory to hybrid systems.

5.1 Logics and Hybrid Systems

Logic-based approaches to hybrid systems are quite common. Several authors have developed new logics or have carried well-known modal and temporal logics over to hybrid systems.

For example, Manna and Pnueli have used *explicit clock temporal logic* [Ost89a, Ost89b, HLP90] to specify properties of hybrid systems [MP93b, MP93a]. In these papers, the authors mainly verify safety conditions of the form $\Box P$. We used similar operators in the previous chapter. Thus we have already shown how to model such conditions algebraically. Another logic that was adapted to hybrid systems is the *duration calculus* (DC) [ZHR91], an interval temporal logic for real-time systems. To describe and model hybrid systems Zhou, Ravn and Hansen extended the original calculus and developed the *extended duration calculus* [ZRH93]. In 1994, Lamport developed the *temporal logic of actions* (TLA). This linear time temporal logic was designed to describe, specify and reason about reactive systems. A complete specification language is given by TLA+ [Lam93, Lam02]. With the goal to investigate modular temporal logic specifications of hybrid systems and constraint-oriented specification structures for hybrid control systems Hermann and Krumm developed a *compositional verification technique* (cTLA) based on TLA [HGK98, HK97]. It supports modular type definitions and composition of processes. Ramadge and Wonham developed the *modular feedback logic* [RW87] for purely discrete event systems. It is closely related to the propositional μ-calculus and not restricted to discrete systems at all. It can easily be adapted to hybrid systems. Work that relates the μ-calculus (and therefore also modular feedback logic) with hybrid systems is given by Davoren in [Dav99]. Finally, differential dynamic logic is yet another logic for hybrid systems [Pla08]. We already embedded this logic into our algebraic setting when presenting the connection between hybrid programs and our algebra in Section 4.1. Other prominent tools in the analysis of concurrent, reactive and hybrid systems are *computation tree logic* (CTL*) [Eme91] — see [AHLP00] and its references for an overview — and *neighbourhood logic* (NL) [ZH04]. We will describe these logics in detail below.

All these temporal and modal logics are well understood, but due to their different notions, syntax and semantics a uniform treatment is mostly not available. Furthermore, their correlation and their relationship to hybrid systems have to be discussed for each logic separately. If an uniform (algebraic) treatment would exist the effort for such an analysis would dramatically decreased. Unfortunately, one rarely finds algebraic treatments of their semantics. We will present algebraic semantics for CTL* and NL. The latter will also give semantics for a number of interval based logics. Algebraic semantics for all remaining logics can probably derived in a similar manner.

First results along these lines concerning CTL* were obtained by von Karger and Berghammer [vK98, vKB98]. But the semantic operators involved were characterised only implicitly. For its sublogic LTL compact closed expressions could be obtained by Desharnais, Möller and Struth in [DMS04a] and, in the framework of fork algebras, by Frías and López Pombo [FLP06]. In the next section we will follow the lines of [MHS06] and provide compact closed semantic expressions for CTL* by using modal

operators in combination with finite and infinite iteration on the basis of lazy quantales: Sets of states and hence the semantics of state formulas can be represented as test elements, while general elements represent the semantics of path formulas. First, our reasoning is purely semantical; later we provide an algebraic interpretation for hybrid systems.

First steps towards an algebraisation of the duration calculus were done in [Höf03, Höf05a]. The duration calculus [ZHR91] itself is an extension of chop-based interval temporal logics like *interval temporal logic* (ITL) [HMM83] and *interval logic* (IL) [Dut95b, Dut95a]. As discussed before all these logics are useful for the specification and verification of safety properties of real-time systems, in particular, of hybrid systems (e.g. [ZRH93]).

However an algebraisation of all these approaches is not necessary since we will present an algebraic treatment for a much richer logic. This generalisation is necessary since ITL, IL and DC have natural limitations due to different aspects. First, it is clear that formulas of these logics cannot express unbounded liveness properties, since their truth value only depends on a given *finite* interval. Furthermore they consider only properties inside that interval and cannot be used for reasoning about properties "outside". Therefore, they are not able to express properties about the perpetual interaction of the system with its environment as in the case of hybrid or reactive systems. Second, notions from real analysis, such as limits, are not expressible in ITL. In order to improve the expressiveness of ITL and DC, they were extended by *infinite intervals* [ZVHX95, Mos00, WX04] and *expanding modalities* [Ven91, Ska94, Pan96, Rab00] that are able to describe behaviour outside the interval under consideration.

Neighbourhood logic (NL) [ZH98] is a first-order interval logic that unifies all interval-based logics. It uses expanding modalities. Its atomic formulas relate time intervals to their (left and right) interval neighbours. It has been shown that the basic unary interval modalities of the *propositional modal logic of time intervals* of Halpern and Shoham [HS91] and the three binary interval modalities (C, T and D) of the *modal logic for chopping intervals* of Venema [Ven91] can be defined using the modalities of NL [GMS04, BGMS07]. Hence NL subsumes those logics. Neighbourhood logic is also used for specifying liveness and fairness of computing systems and for defining notions of real analysis in terms of expanding modalities. Unfortunately NL, as an extension of ITL, is still based on finite intervals and cannot handle infinite intervals. Therefore, although NL is able to reason about past and future behaviour via a universal modality, it cannot reason about unbounded infinite behaviour.

As in the case of CTL* we will provide an algebraic semantics for NL on the basis of lazy quantales: First, we present an algebraic embedding of NL into the algebraic framework of semirings, dealing mainly with the propositional aspects of NL. Then we extend NL from single intervals to sets of intervals, which also paves the way to an

algebraic axiomatisation of NL. In particular, general elements represent the semantics of sets of intervals. The laziness allows us to handle infinite behaviour within the setting of NL[6]. When deriving the algebraic version of NL we obtain further interesting results. For example, some axioms of the original characterisation can be dropped since they are theorems in the algebraic setting. Because of work done by Zhou and Hansen in [ZH98] our extension is also an embedding of interval temporal logic, interval logic, the duration calculus, the propositional modal logic of time intervals and the modal logic for chopping intervals.

Moreover, due to the capability of handling infinite elements, NL and its algebraic counterpart can also be adapted and related to logics like CTL* and hybrid systems. For the latter, the neighbourhood modalities yield some safety and liveness properties. Such properties and similar ones are presented in Section 5.5 when applying these logics to hybrid systems examples.

Since the algebraic treatment of CTL* as well as the one of NL (with all the embedded logics) are first-order, we can again use off-the-shelf theorem-provers to support our calculations.

5.2 Modelling CTL* Algebraically

As discussed in the previous section the temporal logic CTL* is a prominent tool in the analysis of concurrent, reactive and hybrid systems. This section follows the lines of [MHS06] and presents compact closed semantic expressions for CTL* on the basis of lazy quantales. In quantales, sets of states and hence the semantics of state formulas can be represented as test elements in the sense of Kozen [Koz97] (see also Section 3.5), while general elements represent the semantics of path formulas. As a by-product this yields interesting new connections between representations as known from the modal μ-calculus [HKT00] and Kleene/omega algebra.

We first recapitulate the syntax of CTL*. For that, we assume a set Φ of *atomic propositions*. Such propositions stand for atomic facts that may hold in the system under consideration. A possible atomic proposition in the context of the temperature control (cf. Example 4.1.12) is "The temperature is 21 degrees Celsius".

Definition 5.2.1 (CTL* formulas (e.g. [Eme91]))
The language Ψ of CTL* *formulas* over a set Φ of atomic propositions is defined by the grammar

$$\Psi \ ::= \ \bot \mid \Phi \mid \Psi \to \Psi \mid \mathsf{X}\,\Psi \mid \Psi\,\mathsf{U}\,\Psi \mid \mathsf{E}\Psi \ ,$$

[6] To model the original semantics of NL one has to use full quantales.

where X and U are the next-time and the until operators and E is the existential quantifier on paths. The logical connectives $\neg, \wedge, \vee, \mathsf{A}$ are defined, as usual, by $\neg\varphi =_{df} \varphi \to \bot$, $\varphi \wedge \psi =_{df} \neg(\varphi \to \neg\psi)$, $\varphi \vee \psi =_{df} \neg\varphi \to \psi$, $\varphi \leftrightarrow \psi =_{df} (\varphi \to \psi) \wedge (\psi \to \varphi)$ and $\mathsf{A}\varphi =_{df} \neg\,\mathsf{E}\neg\varphi$.[7]

The language of all formulas can be split into two disjoint sets, the state formulas and the path formulas.

Definition 5.2.2 (state formulas, path formulas)
The sublanguages Λ of *state formulas* that denote sets of states and Π of *path formulas* that denote sets of computation traces are given by

$$\begin{aligned} \Lambda &::= \bot \mid \Phi \mid \Lambda \to \Lambda \mid \mathsf{E}\Pi \ , \\ \Pi &::= \Lambda \mid \Pi \to \Pi \mid \mathsf{X}\,\Pi \mid \Pi\,\mathsf{U}\,\Pi \ . \end{aligned}$$

To motivate our algebraic semantics, we recapitulate the standard CTL* semantics formulas. Informally, state formulas are built up from atomic propositions using Boolean connectives and the path quantifiers (E and A). They are used to check wether a property holds in the current state or not. A path (or trace) is a sequence of states. Hence path quantifiers are used to reason about whole paths.

The basic objects (traces) are elements of $\mathcal{P}(\Sigma^\infty - \varepsilon)$. They are sets of finite non-empty or infinite words w over some set Σ of states (cf. Example 3.2.7). The i-th element of w (indices starting with 0) is denoted w_i, and w^i is the trace that results from w by removing its first i elements.

Definition 5.2.3 (standard/concrete semantics of CTL*)
Each atomic proposition $\pi \in \Phi$ is associated with the set $\Sigma_\pi \subseteq \Sigma$ of states for which π is true. The relation $w \models \varphi$ of *satisfaction* of a formula φ by a trace w is defined inductively (e.g. [Eme91]) by

$$\begin{array}{lll} w \not\models \bot \ , & & \\ w \models \pi & \text{iff} & w_0 \in \Sigma_\pi \ , \\ w \models \varphi \to \psi & \text{iff} & w \models \varphi \text{ implies } w \models \psi \ , \\ w \models \mathsf{X}\,\varphi & \text{iff} & \mathrm{ch}\, w^1 \neq \varepsilon \wedge w^1 \models \varphi \ , \\ w \models \varphi\,\mathsf{U}\,\psi & \text{iff} & \exists\, j \geq 0 \,.\, w^j \models \psi \text{ and } \forall\, k < j \,.\, w^k \models \varphi \ , \\ w \models \mathsf{E}\varphi & \text{iff} & \exists\, v \,.\, v_0 = w_0 \text{ and } v \models \varphi \ . \end{array}$$

Informally, the meaning of the temporal operators are as follows: $\mathsf{E}\varphi$ means that there is at least one path starting from the current state where the formula φ is satisfied.

[7] We overload the symbols E and A. They are used to define specifications in Section 4.6 and now for CTL*. However, since these symbols are fairly standard we do not want to change them.

$\mathsf{A}\varphi$ guarantees that φ holds on all paths starting from the current state. $\mathsf{X}\varphi$ postulates that φ holds in the next state. Finally $\varphi\mathsf{U}\psi$ models the until operator; φ has to hold until eventually ψ holds.

The semantics entails that $w \models \neg\varphi$ iff $w \not\models \varphi$ holds. By this we get the following result which will be crucial for the algebraic representation of the next-time operator X.

Lemma 5.2.4 *The proposition that φ holds in the next state is false if and only if the next state does not satisfy φ, i.e.,*

$$w \models \neg\mathsf{X}\varphi \quad \Leftrightarrow \quad w \models \mathsf{X}\neg\varphi$$

for an arbitrary trace w with $w^1 \neq \varepsilon$.

Proof. By the assumption, the definition of $w \models \mathsf{X}\varphi$ reduces to $w^1 \models \varphi$. Then, the above definitions immediately imply the claim:

$$w \models \neg\mathsf{X}\varphi \Leftrightarrow w \not\models \mathsf{X}\varphi \Leftrightarrow w^1 \not\models \varphi \Leftrightarrow w^1 \models \neg\varphi \Leftrightarrow w \models \mathsf{X}\neg\varphi \ .$$

□

From this semantics one can extract a set-based one by assigning to each formula φ the set $[\![\varphi]\!] =_{df} \{w \mid w \models \varphi\}$ of paths that satisfy it.

As we have seen in Example 3.2.7 sets of traces form a Boolean lazy quantale, hence this is the basis of the algebraic semantics. We now give an algebraic interpretation of CTL^* over such an algebraic structure S. To save some notation we identify the set of all atomic propositions Φ with $\mathsf{test}(S)$. Moreover, we fix an element n (n standing for "next") that represents the transition system underlying the logic. The precise requirements for n will be discussed below. Then the concrete semantics above generalises to a function that maps sets of paths to semiring elements.

Theorem 5.2.5 (algebraic CTL^* semantics) *For an arbitrary Boolean lazy quantale, the standard semantics of CTL^* generalises to a function $[\![\_]\!] : \Psi \to S$:*

$$\begin{aligned}
[\![\bot]\!] &= 0 \ , \\
[\![p]\!] &= p \cdot \top \ , \\
[\![\varphi \to \psi]\!] &= \overline{[\![\varphi]\!]} + [\![\psi]\!] \ , \\
[\![\mathsf{X}\,\varphi]\!] &= \mathsf{n} \cdot [\![\varphi]\!] \ , \\
[\![\varphi\,\mathsf{U}\,\psi]\!] &= \bigsqcup_{j \geq 0} (\mathsf{n}^j \cdot [\![\psi]\!] \sqcap \bigsqcap_{k<j} \mathsf{n}^k \cdot [\![\varphi]\!]) \ , \\
[\![\mathsf{E}\varphi]\!] &= \lceil [\![\varphi]\!] \cdot \top \ .
\end{aligned}$$

As we will see in the next section, formulas like $\lceil [\![\varphi]\!] \cdot \top$ will also occur in the algebraic version of neighbourhood logic. Hence both logics provide a common part and allow cross-reasoning. Moreover, the algebraic semantics of CTL* is essentially based on Boolean complement and the existence of a greatest element. To use the above algebraic semantics freely and since the algebra of hybrid systems PRO forms such a Boolean lazy quantale, we assume for the remainder of this section that S is such a quantale.

Corollary 5.2.6 *Using Theorem 5.2.5, it is straightforward to check that*

$$[\![\varphi \vee \psi]\!] = [\![\varphi]\!] + [\![\psi]\!] , \qquad [\![\varphi \wedge \psi]\!] = [\![\varphi]\!] \sqcap [\![\psi]\!] , \qquad [\![\neg\varphi]\!] = \overline{[\![\varphi]\!]} .$$

Given a set Σ of states, over the lazy quantale $\mathsf{STR}(\Sigma)$ (see Example 3.2.7) this semantics coincides with the standard semantics of CTL* (Definition 5.2.3). Another important check of the adequacy of our definitions is provided by the following theorem. The restriction on n mentioned in the assumption will be discussed later.

Theorem 5.2.7 *Assume that left multiplication with* n *distributes through meets. Then the element* $[\![\varphi \,\mathsf{U}\, \psi]\!]$ *is the least fixpoint* μf *of the function*

$$f(y) =_{df} [\![\psi]\!] + ([\![\varphi]\!] \sqcap \mathsf{n} \cdot y) .$$

Proof. Since in a Boolean quantale multiplication and binary meet preserve arbitrary joins, f preserves arbitrary joins, too, and hence is continuous. So by Kleene's fixpoint theorem $\mu f = \bigsqcup_{j \geq 0} f^j(0)$. The claim is immediate from $f^i(0) = \bigsqcup_{j \leq i} (\mathsf{n}^j \cdot [\![\psi]\!] \sqcap \prod_{k<j} \mathsf{n}^k \cdot [\![\varphi]\!])$, which can be shown by a straightforward induction. □

Definition 5.2.8 (derived temporal operators for CTL*)
As usual in temporal logics, we can define more temporal operators from U and E by

$$\mathsf{A}\varphi =_{df} \neg\mathsf{E}\neg\varphi , \qquad \mathsf{F}\varphi =_{df} \top\mathsf{U}\varphi , \qquad \mathsf{G}\varphi =_{df} \neg\mathsf{F}\neg\varphi .^{8}$$

We have already discussed the meaning of A. Informally, $\mathsf{F}\varphi$ describes the situation where φ has to hold eventually, whereas for $\mathsf{G}\varphi$ the formula φ has to hold on the entire subsequent path.

Corollary 5.2.9 *By the above results there is a closed representation of* F*, namely*

$$[\![\mathsf{F}\varphi]\!] = \mathsf{n}^* \cdot [\![\varphi]\!] .$$

[8] Here, we overload symbols again. Depending on the situation F denotes the greatest purely finite element or the eventually-operator of CTL*.

To find suitable requirements on n in the algebraic setting, we consider Lemma 5.2.4 that has to be satisfied. To fulfil this lemma, we need to have for all formulas φ and their semantical values $b =_{df} [\![\varphi]\!]$,

$$\overline{\mathsf{n} \cdot b} = [\![\neg \mathsf{X} \varphi]\!] = [\![\mathsf{X} \neg \varphi]\!] = \mathsf{n} \cdot \overline{b} \; . \tag{5.1}$$

This semantic property can equivalently be characterised as follows.

Lemma 5.2.10 *Consider a Boolean weak quantale S and $\mathsf{n} \in S$ such that $\mathsf{n} \cdot 0 = 0$. (n must be a purely finite element.)*

1. $\forall b \in S : \mathsf{n} \cdot \overline{b} \leq \overline{\mathsf{n} \cdot b} \Leftrightarrow \forall b, c \in S : \mathsf{n} \cdot (b \sqcap c) = \mathsf{n} \cdot b \sqcap \mathsf{n} \cdot c$.
2. $\forall b \in S : \overline{\mathsf{n} \cdot b} \leq \mathsf{n} \cdot \overline{b} \Leftrightarrow \mathsf{n} \cdot \top = \top \Leftrightarrow \mathsf{n}^\omega = \top$.

The proof can be found in the Appendix.

In relation algebra REL, the special case $\mathsf{n} \cdot \overline{1} \leq \overline{n}$ of the property in Part (1) characterises n as a partial function (e.g. [SS93]). But in general quantales the special and the general case are not equivalent [DM01]. Moreover, again from [DM01], we know that in Boolean lazy quantales such as WOR, STR and PRO an element n is left-distributive over meet if and only if it is prefix-free, i.e. if no member of n is a prefix of another member. This holds in particular if all words in n have equal length, which is the case if n models a transition relation and hence consists only of words of length 2. The equivalent condition $\forall b \,.\, \mathsf{n} \cdot b \sqcap \mathsf{n} \cdot \overline{b} = 0$ was used in the computation calculus of Dijkstra [Dij00].

But what about Part (2)? Only rarely will a quantale be "generated" by an element n in the sense that $\mathsf{n}^\omega = \top$. The solution is to choose a left-distributive element n and restrict the set of semantical values to the subset $\mathrm{SEM}(\mathsf{n}) =_{df} \{b : b \leq \mathsf{n}^\omega\}$, taking complements relative to n^ω. This set is clearly closed under $+$ and $\sqcap$ and under prefixing by n, since by isotony

$$\mathsf{n} \cdot b \leq \mathsf{n} \cdot \mathsf{n}^\omega = \mathsf{n}^\omega \; .$$

Finally, it also contains all elements $p \cdot \mathsf{n}^\omega$ with $p \in \mathsf{test}(S)$, since $p \leq 1$. Hence the above semantics is well-defined in $\mathrm{SEM}(\mathsf{n})$ if we replace $\top$ by n^ω. In Section 5.5 we will discuss if there are such elements in PRO and if it is useful to use the next-operator when reasoning about hybrid systems. The assumption that n has to be purely finite is quite clear since a possible infinite element n cannot guarantee that the next "state" is reached. However, except from X the other temporal operators and logical connectives are useful for hybrid systems.

Before turning to neighbourhood logic and, later on, to examples and applications for hybrid systems, we will have a short look at the semantics of state formulas. In

particular, we show, next to some other properties, that the semantics of each state formula has the special form of a test ideal and hence directly corresponds to a test, i.e., an abstract representation of a set of states. This is the key to a simplified CTL semantics in [MHS06].

Theorem 5.2.11 (semantics of state formulas) *Let φ be a state formula.*

1. *$[\![\varphi]\!]$ is a test ideal, i.e., $[\![\varphi]\!] = [\![\varphi]\!] \cdot \top$, and hence, by Lemma 3.6.3(4), $[\![\varphi]\!] = \lceil([\![\varphi]\!]) \cdot \top = \lceil([\![\varphi]\!] \cdot \top)$.*
2. *$[\![\mathsf{E}\varphi]\!] = [\![\varphi]\!]$.*
3. *$[\![\mathsf{A}\varphi]\!] = \neg\lceil(\overline{[\![\varphi]\!]}) \cdot \top$.*

Proof.

1. The proof is by induction on the structure of φ.
   - For $\bot$ and $p \in \mathsf{test}(S)$ this is immediate from the definition.
   - Assume that the claim already holds for state formulas φ and ψ. We calculate, using the definitions, the induction hypothesis, Corollary 3.5.4, distributivity and the definitions again,
   $$\begin{aligned}[\![\varphi \to \psi]\!] &= \overline{[\![\varphi]\!]} + [\![\psi]\!] = \overline{\lceil[\![\varphi]\!] \cdot \top} + \lceil[\![\psi]\!] \cdot \top = \neg\lceil[\![\varphi]\!] \cdot \top + \lceil[\![\psi]\!] \cdot \top \\ &= (\neg\lceil[\![\varphi]\!] + \lceil[\![\psi]\!]) \cdot \top = (\lceil[\![\varphi]\!] \to \lceil[\![\psi]\!]) \cdot \top.\end{aligned}$$
   - For $\mathsf{E}\varphi$ the claim is immediate from the definition.
2. Immediate from (1) and the definition of $[\![\mathsf{E}\varphi]\!]$.
3. Similar to (2). □

Moreover, state formulas are closed under $\neg, \wedge, \vee$ and A.

To conclude this chapter we list a couple of properties and algebraic expressions for CTL*-formulas. In particular, we derive some properties of U and its relatives for state formulas. Moreover we deal with E, A and their connections. Most of the proofs are straightforward calculations; some are listed in Appendix B.

Lemma 5.2.12 *Let φ, ψ be state formulas of CTL* and $p \cdot \top =_{df} [\![\varphi]\!]$, $q \cdot \top =_{df} [\![\psi]\!]$.*

1. *$[\![\varphi \mathsf{U} \psi]\!] = (p \cdot \mathsf{n})^* \cdot q \cdot \top = ([\![\varphi]\!] \sqcap \mathsf{n})^* \cdot [\![\psi]\!]$. The case $p = 1$ yields again Corollary 5.2.9.*
2. *$[\![\mathsf{G}\varphi]\!] = (p \cdot \mathsf{n})^\omega = ([\![\varphi]\!] \sqcap \mathsf{n})^\omega$.*
   Hence we have the shunting rule $(p \cdot \mathsf{n})^\omega = \overline{\mathsf{n}^* \cdot \neg p \cdot \top}$.

Lemma 5.2.13 *For atomic proposition* $p \in \mathsf{test}(S)$,

$$\begin{array}{rcl@{\qquad}rcl} [\![\mathsf{A}\bot]\!] & = & 0\ , & [\![\mathsf{A}\top]\!] & = & \top\ , \\ [\![\mathsf{A}(p \vee \varphi)]\!] & = & p + [\![\mathsf{A}\varphi]\!]\ , & [\![\mathsf{A}(p \wedge \varphi)]\!] & = & p \cdot [\![\mathsf{A}\varphi]\!]\ . \end{array}$$

Lemma 5.2.14 *The equation* $[\![\mathsf{EX}\varphi]\!] = [\![\mathsf{EXE}\varphi]\!]$ *holds. Moreover,*

$$[\![\mathsf{EX}\top]\!] = \top \Leftrightarrow \ulcorner \mathsf{n} = 1 \Leftrightarrow \mathsf{n} \text{ } \textit{total}\ .$$

Proof. By the definitions, properties of domain, Equation (d3) and the definitions again,

$$[\![\mathsf{EXE}\varphi]\!] = \ulcorner(\mathsf{n} \cdot \ulcorner[\![\varphi]\!] \cdot \top) \cdot \top = \ulcorner(\mathsf{n} \cdot \ulcorner[\![\varphi]\!]) \cdot \top = \ulcorner(\mathsf{n} \cdot [\![\varphi]\!]) \cdot \top = [\![\mathsf{EX}\varphi]\!]\ .$$

The second claim follows by Lemma 3.5.5(1), since $[\![\mathsf{EX}\top]\!] = \ulcorner(\mathsf{n} \cdot \top) \cdot \top = \ulcorner \mathsf{n} \cdot \top$. □

Lemma 5.2.15 EX *and* AX *are de Morgan duals, i.e.,* $[\![\mathsf{AX}\varphi]\!] = [\![\neg\mathsf{EX}\neg\varphi]\!]$. *By Lemma 5.2.14 this implies* $[\![\mathsf{AX}\varphi]\!] = [\![\mathsf{AXA}\varphi]\!]$.

For a number of applications the sublogic CTL or LTL of CTL^* suffices. CTL, a sublogic of CTL^* where only path formulas of a restricted form are allowed, can be modelled in plain Kleene algebra. The logic LTL is the fragment of CTL^* which contains no path quantifier except A. Moreover, this path quantifier may only occur at the outermost level. This can also be modelled by pure Kleene algebra. To achieve this goal one can define suitable mappings that, for the CTL and LTL formulas, transform their general CTL^* semantics into simplified versions in omega-regular form (cf. [MHS06]). Since for verification tasks of hybrid systems the full semantics of CTL^* is useful, we skip these sublogics.

5.3 Modelling Neighbourhood Logic Algebraically

Before returning to hybrid system analysis, we will provide compact closed semantics for the neighbourhood logic of Zhou and Hansen [ZH98]. The embedding of NL into the setting of idempotent semiring was first done in [Höf07]. Later on, we expanded this approach to lazy semirings [HM06, HM08]

As in the case of CTL^*, we briefly give an overview over the original semantics.

Neighbourhood logic (NL) is a logical formalism for reasoning about liveness and fairness properties in the framework of finite intervals. It is mainly based on interval temporal logic (ITL) [HMM83, HS91] and provides the possibility to "look" beyond

the current interval. This additional feature allows expressing unbounded liveness properties, like "eventually there will be a time interval, where φ holds" and "φ will hold infinitely often in the future", which are not expressible in the setting of ITL. For the temperature control (Example 4.1.12) this may be "Sometimes, the temperature will be exactly 21 degrees Celsius".

Definition 5.3.1 (vocabulary of NL)
The vocabulary of NL consists of time-independent *global variable symbols* x, time-dependent *temporal variable symbols* v, time-independent *global function symbols* f^n, time-independent *temporal propositional letters* X and time-independent *global relation symbols* G^n (for certain arities $n \in \mathbb{N}$). There are two special global variables true and false and a special temporal variable ℓ which denotes the length of the interval under consideration.

Definition 5.3.2 (terms and formulas of neighbourhood logic)
The languages Θ of NL *terms* and Φ of NL *formulas* over the above vocabulary are defined by the semi-formal grammar

$$\Theta ::= x \mid v \mid f^n(\underbrace{\Theta, \ldots, \Theta}_{n}) \ ,$$

$$\Phi ::= X \mid G^n(\underbrace{\Theta, \ldots, \Theta}_{n}) \mid \Phi \wedge \Phi \mid \neg\Phi \mid (\exists x)\Phi \mid \Diamond_l\Phi \mid \Diamond_r\Phi \ .$$

The logical connectives are again defined as usual; the dual for $(\exists x)$ is defined as $(\forall x)\varphi =_{df} \neg(\exists x)\neg\varphi$. Furthermore, we define duals of the diamond operators $\Diamond_l$ and $\Diamond_r$ in the standard way by $\Box_l\varphi =_{df} \neg\Diamond_l\neg\varphi$ and $\Box_r\varphi =_{df} \neg\Diamond_r\neg\varphi$.

The original semantics of NL is based on the arithmetic of real numbers (see [ZH98]). It is well known that NL allows an arbitrary cancellative commutative group as its time domain. However, since we want to derive an abstract algebraic version of NL later on, we focus on the intuition of NL which is more directly presented with real numbers.

Its basic objects are real-valued intervals $[y, z]$ with $y \leq z$ and $y, z \in \mathbb{R}$. The meanings of f^n and G^n are straightforwardly given by functions $\underline{f}^n \in \mathbb{R}^n \to \mathbb{R}$ and $\underline{G}^n \in \mathbb{R}^n \to \{\mathsf{true}, \mathsf{false}\}$. The meanings of global variables are given by a *value assignment* $\mathcal{V}$, a function that assigns a real number $x^{\mathcal{V}}$ to each global variable x. The meanings of temporal variables and propositional letters are given by an *interpretation* $\mathcal{J}$, a function that associates a real-valued interval function $v^{\mathcal{J}}$ with each temporal variable v and a truth-valued interval function $X^{\mathcal{J}}$ with each propositional letter X. For example, the interpretation of the temporal variable ℓ is

$$\ell^{\mathcal{J}}([y, z]) \ = \ z - y \ .$$

The semantics of terms and the logical symbols other than $\diamondsuit_l$ and $\diamondsuit_r$ is standard.

Definition 5.3.3 (standard semantics of NL)
The semantics $\theta^{\mathcal{J},\mathcal{V}}$ of a term θ with respect to an interpretation $\mathcal{J}$ and value assignment $\mathcal{V}$ can be given inductively as an interval function:

$$\begin{array}{ll} x^{\mathcal{J},\mathcal{V}}([y,z]) & =_{df}\ \mathcal{V}(x)\ , \\ v^{\mathcal{J},\mathcal{V}}([y,z]) & =_{df}\ v^{\mathcal{J}}([y,z])\ , \\ f^n(\theta_1 \ldots \theta_n)^{\mathcal{J},\mathcal{V}}([y,z]) & =_{df}\ \underline{f}^n(c_1 \ldots c_n)\ , \end{array}$$

where $c_i = \theta_i^{\mathcal{J},\mathcal{V}}([y,z]), i = 1 \ldots n$. Further on, one inductively defines when a formula φ *holds* for an interpretation $\mathcal{J}$, a value assignment $\mathcal{V}$ and an interval $[y,z]$, in signs $[y,z] \models_{\mathcal{J},\mathcal{V}} \varphi$:

$$\begin{array}{lll} [y,z] \models_{\mathcal{J},\mathcal{V}} X & \text{iff} & X^{\mathcal{J}}([y,z]) = \mathsf{true}\ , \\ [y,z] \models_{\mathcal{J},\mathcal{V}} G^n(\theta_1,\ldots,\theta_n) & \text{iff} & \underline{G}^n(c_1,\ldots,c_n) = \mathsf{true}, \\ & & \text{where } c_i = \theta_i^{\mathcal{J},\mathcal{V}}([y,z]),\ i = 1 \ldots n\ , \\ [y,z] \models_{\mathcal{J},\mathcal{V}} \neg\varphi & \text{iff} & [y,z] \not\models_{\mathcal{J},\mathcal{V}} \varphi\ , \\ [y,z] \models_{\mathcal{J},\mathcal{V}} \varphi \vee \psi & \text{iff} & [y,z] \models_{\mathcal{J},\mathcal{V}} \varphi \ \text{ or } \ [y,z] \models_{\mathcal{J},\mathcal{V}} \psi\ , \\ [y,z] \models_{\mathcal{J},\mathcal{V}} (\exists x)\varphi & \text{iff} & [y,z] \models_{\mathcal{J},\mathcal{V}'} \varphi \text{ for some } \mathcal{V}' \text{ that agrees with } \mathcal{V}\ , \\ & & \text{for all global variables } u \neq x \\ [y,z] \models_{\mathcal{J},\mathcal{V}} \diamondsuit_l\varphi & \text{iff} & \exists\delta \geq 0 : [y-\delta, y] \models_{\mathcal{J},\mathcal{V}} \varphi\ , \\ [y,z] \models_{\mathcal{J},\mathcal{V}} \diamondsuit_r\varphi & \text{iff} & \exists\delta \geq 0 : [z, z+\delta] \models_{\mathcal{J},\mathcal{V}} \varphi\ . \end{array}$$

For further details we refer to the original work of Barua, Hansen, Roy and Zhou [BRZ00, ZH04]. Intuitively, $\diamondsuit_l$ and $\diamondsuit_r$ allow reasoning about *left* and *right* neighbourhoods of a given interval. This behaviour is shown in Figure 5.1.

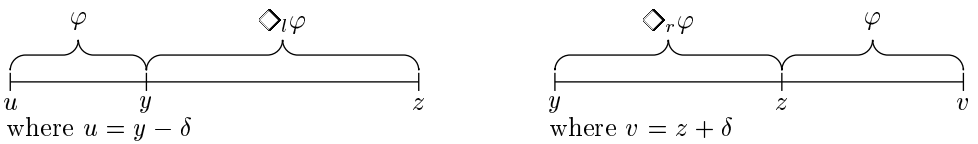


Figure 5.1: Left and right neighbouhoods of an interval $[y,z]$

By definition, $\diamondsuit_l\varphi$ will hold for an interval that has an interval on the *left* where φ holds. Symmetrically, $\diamondsuit_r\varphi$ holds for an interval having a *right* neighbour interval where φ holds.

In interval (temporal) logics the *chop operator* plays a crucial rôle. This binary interval modality can be interpreted as the operation of "chopping" an interval into two parts.

Definition 5.3.4 (semantics of the chop operator)
The semantics of the chop operator $\frown$ is given by

$$[y,z] \models_{\mathcal{J},\mathcal{V}} \varphi \frown \psi \;\Leftrightarrow\; \exists m : y \leq m \leq z \;\wedge\; [y,m] \models_{\mathcal{J},\mathcal{V}} \varphi \;\wedge\; [m,z] \models_{\mathcal{J},\mathcal{V}} \psi \;.$$

It is well known that the chop operator cannot be derived from the basic unary modalities in a propositional logic like ITL [Ven90, BGMS07], but it is expressible in NL and therefore ITL is subsumed [ZH98].

Theorem 5.3.5 *In neighbourhood logic the chop operator is expressed by*

$$\varphi \frown \psi \;\Leftrightarrow\; \exists x, y : \Big((\ell = x + y) \;\wedge\; \Diamond_l \Diamond_r \big((\ell = x) \wedge \varphi \wedge \Diamond_r ((\ell = y) \wedge \psi) \big) \Big) \;.$$

The proof can be found in [ZH04]. In Chapter 4 we introduced an algebraic chop operator. Later on we will show that both notions coincide.

There are various kinds of interval temporal logics in the literature, both propositional ([HS91, HMM83, Ven91]) and first-order ([Dut95b]). Most of these logics are subsumed by NL (e.g. [ZH04, Theorem 11.1]).

Since NL is a logic based on *finite* intervals, an infinite behaviour can therefore only be approximated by finite prefixes of the respective infinite interval. As we have seen in Section 4.4 such an approximation does not work for arbitrary processes. Shortly we will discuss a possibility to introduce infinite intervals into NL. An extension of ITL to discrete-time infinite intervals has already been given in [Mos00].

Let us now turn to the algebraic semantics of neighbourhood logic. For that we assume a fixed interpretation $\mathcal{J}$ and value assignment $\mathcal{V}$ and abbreviate $\models_{\mathcal{J},\mathcal{V}}$ by just $\models$. Similar to CTL*, we assign to each formula φ the set $\mathbb{I}_\varphi \;=_{df}\; \{[y,z] \,|\, [y,z] \models \varphi\}$ of all intervals where φ holds. The sets $\mathbb{I}_\varphi$ of intervals will be the elements of our algebraic structure.

Interval composition is defined as usual:

$$[y_1,z_1] \,;\, [y_2,z_2] \;=_{df}\; \begin{cases} [y_1,z_2] & \text{if } z_1 = y_2 \\ \text{undefined} & \text{otherwise} \end{cases} .$$

Sets of (finite) intervals form a Boolean semiring when using the lifted interval composition as product. Hence $\mathsf{INT} \;=_{df}\; (\mathcal{P}(\mathbb{I}), \cup, ;, \emptyset, \mathbb{1})$ will be the basis of the algebraic semantics where $\mathbb{I}$ is the set of all intervals over $\mathbb{R}$ and $\mathbb{1}$ is the set of all one-point intervals. The proof is either by straightforward calculations (e.g., [Höf05b]) or by relating it to the algebra of hybrid systems PRO.

Theorem 5.3.6 INT *is isomorphic to a subalgebra of* PRO.

Proof. Let $[y,z]$ be an interval of $\mathbb{R}$ and $V = \mathbb{R}$. At first glance one might think to identify each interval $[y,z]$ with a constant trajectory, that means $[y,z] \cong \{(z-y,g) \,|\, g(x) = v,\, x \in [0,z-y]\}$, where $v \in V$ is an arbitrary but fixed value. However, the problem is that in PRO all trajectories are normalised and start at time 0. Therefore using the above connection would loose the composition conditions. To overcome this difficulty, we take the length of an interval as its duration and use the identity function. This construction guarantees that only those intervals can be composed where the end point of the first equals the starting point of the second. In detail we use the following construction:

$$[y,z] \cong \{(z-y,g) \,|\, g(x) = y + x,\, x \in [0,z-y]\} \;.$$

By this, each interval is isomorphic to a process with finite duration. The claim then follows immediately. □

We can determine the domain and codomain of INT. For a set A of finite intervals[9] we have $\ulcorner A = \{[y,y] : \exists\, z : [y,z] \in A\}$ and $A^{\urcorner} = \{[z,z] : \exists\, y : [y,z] \in A\}$.

Obviously, temporal and global variables as well as propositional letters can be used to construct such sets of intervals. For example, using the temporal variable ℓ, we can characterise all intervals of length d by $\mathbb{I}_{\ell=d}$. The embedding of other NL formulas is then straightforward.

Theorem 5.3.7 (algebraic NL semantics) *For* INT*, the standard semantics of* NL *generalises to a subset relation.*

$$\begin{array}{lcl}
[y,z] \models \varphi & \Leftrightarrow & [y,z] \in \mathbb{I}_\varphi \;, \\
[y,z] \models \neg\varphi & \Leftrightarrow & [y,z] \in \overline{\mathbb{I}_\varphi} \;, \\
[y,z] \models \varphi \vee \psi & \Leftrightarrow & [y,z] \in \mathbb{I}_{\varphi\vee\psi} = \mathbb{I}_\varphi \cup \mathbb{I}_\psi \;, \\
[y,z] \models \Diamond_l\varphi & \Leftrightarrow & [y,y] \in \mathbb{I}_\varphi^{\urcorner} \;, \\
[y,z] \models \Diamond_r\varphi & \Leftrightarrow & [z,z] \in {}^{\ulcorner}\mathbb{I}_\varphi \;.
\end{array}$$

We lift the validity assertion to sets of intervals by setting, for $A \subseteq \mathbb{I}$*,*

$$A \models \varphi \;\Leftrightarrow\; \forall [y,z] \in A : [y,z] \models \varphi \;\Leftrightarrow\; A \subseteq \mathbb{I}_\varphi \;.$$

In particular, $A \models \Diamond_l\varphi \Leftrightarrow {}^{\ulcorner}A \subseteq \mathbb{I}_\varphi^{\urcorner}$ *and* $A \models \Diamond_r\varphi \Leftrightarrow A^{\urcorner} \subseteq {}^{\ulcorner}\mathbb{I}_\varphi$*.*

Proof. The first three lines are straightforward. Furthermore we get

$$\begin{array}{l}
[y_1,z_1] \text{ is a left neighbour of } [y_2,z_2] \\
\quad \Leftrightarrow\; [y_2,z_2] \text{ is a right neighbour of } [y_1,z_1] \\
\quad \Leftrightarrow\; [y_1,z_1] \,;\, [y_2,z_2] \text{ is defined} \\
\quad \Leftrightarrow\; z_1 = y_2
\end{array}$$

[9] Remember that we assumed only finite intervals in INT.

and therefore we calculate for $\Diamond_l \varphi$

$$\begin{aligned}
[y,z] \models \Diamond_l \varphi &\Leftrightarrow \exists\, [u_1,u_2] \in \mathbb{I}_\varphi : [u_1,u_2]\,;[y,z] \text{ is defined} \\
&\Leftrightarrow \exists\, [u_1,u_2] \in \mathbb{I}_\varphi : y = u_2 \\
&\Leftrightarrow \exists\, u_1 : [u_1,y] \in \mathbb{I}_\varphi \\
&\Leftrightarrow [y,y] \in \mathbb{I}_\varphi^\urcorner \,.
\end{aligned}$$

$A \models \Diamond_l \varphi \Leftrightarrow {}^\ulcorner A \subseteq \mathbb{I}_\varphi^\urcorner$ then follows by disjunctivity of domain and ${}^\ulcorner\{[y,z]\} = \{[y,y]\}$. The claims concerning $\Diamond_r \varphi$ can be proved similar. □

As a first result of the algebraisation we note that at least one of the eight axioms postulated in [ZH98] can be dropped, since it is a theorem in domain semirings.

Theorem 5.3.8 *On single intervals,* $\Diamond(\varphi \vee \psi) \Leftrightarrow \Diamond\varphi \vee \Diamond\psi$*, where* $\Diamond$ *is* $\Diamond_l$ *or* $\Diamond_r$*. Hence Axiom 4 of [ZH98], which postulates the distributivity of* $\Diamond$ *over disjunction, is now a conclusion.*

Proof. Using Theorem 5.3.7, ${}^\ulcorner\{[y,z]\} = \{[y,y]\}$ and additivity of codomain, we get

$$\begin{aligned}
[y,z] \models \Diamond_l \varphi \vee \Diamond_l \psi &\Leftrightarrow {}^\ulcorner\{[y,z]\} \subseteq \mathbb{I}_\varphi^\urcorner \vee {}^\ulcorner\{[y,z]\} \subseteq \mathbb{I}_\psi^\urcorner \\
&\Leftrightarrow {}^\ulcorner\{[y,z]\} \subseteq \mathbb{I}_\varphi^\urcorner \cup \mathbb{I}_\psi^\urcorner \\
&\Leftrightarrow {}^\ulcorner\{[y,z]\} \subseteq (\mathbb{I}_\varphi \cup \mathbb{I}_\varphi)^\urcorner = (\mathbb{I}_{\varphi\vee\psi})^\urcorner \\
&\Leftrightarrow [y,z] \models \Diamond_l(\varphi \vee \psi) \,.
\end{aligned}$$

The proof of distributivity of $\Diamond_r$ is similar. □

More precisely, the corresponding logical part of Theorem 5.3.8 splits into two parts. The first one, $\Diamond(\varphi \vee \psi) \Rightarrow \Diamond\varphi \vee \Diamond\psi$ is also a consequence of the axioms M and K for modal logic (e.g., [HC96]), as stated in [HM08]. The second, $\Diamond\varphi \vee \Diamond\psi \Rightarrow \Diamond(\varphi \vee \psi)$ is already a theorem in NL (e.g. Theorem NL3 in [ZH04]).

More simplifications can be achieved by this algebraic embedding. They are listed in [HM06, HM08]. In [HM08] we further show that the temporal-based interval logic introduced by Allen in [All83, AH85] is also subsumed.

Next we discuss the box operators $\Box_l \varphi$ and $\Box_r \varphi$ of Zhou and Hansen in the setting of modal semirings. The meaning of $\Box_l \varphi$ and $\Box_r \varphi$ is the following:

$$\begin{aligned}
[y,z] \models \Box_l \varphi &\Leftrightarrow i \models \varphi \text{ for all left neighbour intervals } i \text{ of } [y,z] \,, \\
[y,z] \models \Box_r \varphi &\Leftrightarrow i \models \varphi \text{ for all right neighbour intervals } i \text{ of } [y,z] \,.
\end{aligned}$$

Again we start with the pointwise characterisation of the boxes in INT.

Since $\lceil\{[y,z]\} = \{[y,y]\}$ is a singleton set,

$$\begin{aligned}
[y,z] \models \Box_l \varphi &\Leftrightarrow [y,z] \models \neg\Diamond_l\neg\varphi \\
&\Leftrightarrow \lceil\{[y,z]\} \not\subseteq (\mathbb{I}_{\neg\varphi})\rceil \\
&\Leftrightarrow \lceil\{[y,z]\} \subseteq \neg(\mathbb{I}_{\neg\varphi})\rceil \\
&\Leftrightarrow (\mathbb{I}_{\neg\varphi})\rceil ; \lceil\{[y,z]\} \subseteq \emptyset \ .
\end{aligned}$$

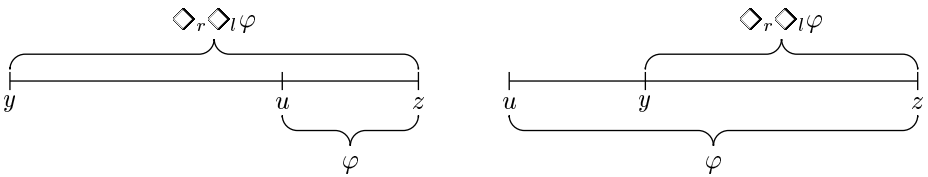
Note that the symbol $\neg$ is overloaded and used in two different contexts; on the one hand it is the logical negation of NL and on the other hand it denotes the complement of tests.

Since $\mathbb{I}_{\neg\varphi}$ characterises the set of all intervals where φ does not hold, it is the same as $\overline{\mathbb{I}_\varphi}$ using the complement function of Boolean semirings and the definition of $\mathbb{I}_\varphi$. Using the same generalisation as above we get the following lemma.

Lemma 5.3.9 *Assume a set of intervals $A \subseteq \mathbb{I}$.*

$$\begin{aligned}
A \models \Box_l \varphi &\Leftrightarrow (\mathbb{I}_{\neg\varphi})\rceil ; \lceil A \subseteq \emptyset \ , \\
A \models \Box_r \varphi &\Leftrightarrow A\rceil ; \lceil(\mathbb{I}_{\neg\varphi}) \subseteq \emptyset \ .
\end{aligned}$$

In [ZH04], the authors introduce additional neighbourhood modalities for NL which are given by composing the basic modalities $\Diamond_l$ and $\Diamond_r$. We show that they are again diamonds closely related to $\Diamond_l$ and $\Diamond_r$. The meaning of $\Diamond_r\Diamond_l\varphi$ is shown in Figure 5.2.



where $u = z - \delta$

Figure 5.2: Nested neigbourhood modalities

In this case, $[u,z]$ is a postfix of $[y,z]$, or $[y,z]$ is a postfix of $[u,z]$. These nested diamond operators are closely related to the modal operators $E, \overline{E}, B$ and $\overline{B}$ of the logic defined in [HS91]. For details of the relationship see [ZH04].

In contrast to the simple neighbourhood operators where some starting points have to be equal to some end points of sets of intervals, here only end points occur. The

end points of $\Diamond_r\Diamond_l\varphi$ have to form a subset of the ones of φ. Using the (co)domain interpretation (cf. proof of Theorem 5.3.7),

$$\begin{aligned}
[y,z] \models \Diamond_r\Diamond_l\varphi &\Leftrightarrow \{[y,z]\}^\urcorner \subseteq \ulcorner\left(\mathbb{I}_{\Diamond_l\varphi}\right)\\
&\Leftrightarrow \{[y,z]\}^\urcorner \subseteq \ulcorner\{[u,v] : \ulcorner\{[u,v]\} \subseteq \mathbb{I}_\varphi^\urcorner\}\\
&\Leftrightarrow \{[y,z]\}^\urcorner \subseteq \{[u,u] : [u,u] \in \mathbb{I}_\varphi^\urcorner\}\\
&\Leftrightarrow \{[y,z]\}^\urcorner \subseteq \mathbb{I}_\varphi^\urcorner\ .
\end{aligned}$$

We can derive a similar expression for $\Diamond_l\Diamond_r\varphi$ as $\ulcorner\{[y,z]\} \subseteq \ulcorner\mathbb{I}_\varphi$. In our setting the characterisation of $\Diamond_r\Diamond_l\varphi$ and $\Diamond_l\Diamond_r\varphi$ is no more complicated than that of the single neighbourhood modalities. The four neighbourhood operators ($\Diamond_l$, $\Diamond_r$, $\Diamond_l\Diamond_r$, $\Diamond_r\Diamond_l$) represent all combinations for comparing domain and codomain and therefore motivate the definition for neighbours in the general setting of lazy semirings in the next section.

A severe limitation of NL and also the presented algebraic setting is that they cannot handle *infinite* intervals, which is necessary to describe infinite behaviour. For example, including infinite intervals allows expressing properties about perpetual interaction of a system with its environment as in the case of hybrid or reactive systems. Therefore we present an extension of NL in which this can be modelled. Instead of finite intervals we now use intervals with finite and infinite length.

Similar to composition of trajectories (cf. Section 4.1), we define composition for finite and infinite intervals. The composition of two finite intervals is defined as before. The missing cases are $[y_1,z_1]\,;[y_2,\infty[=_{df} [y_1,\infty[$ if $z_1 = y_2$ and undefined otherwise, and, $[y,\infty[\,;i =_{df} [y,\infty[$ for any interval i. Intuitively, the latter case describes the situation that the second interval is never reached, since the first one is already infinite.

Using the same arguments as above, it is straightforward to see that this extended structure forms again a lazy semiring and is also isomorphic to a subalgebra of PRO. In particular, the structure is right-distributive.

Due to the algebraic structure, there is nothing more to do to get a neighbourhood logic with possible infinite intervals. Nevertheless we will give a short example.

By the definition of interval (trajectory) composition each interval can be composed to an infinite one from the right. For a single infinite interval we have

$$\begin{aligned}
\{[y,\infty[\} \models \Diamond_r\varphi &\Leftrightarrow \exists\,[u_1,u_2] \in \mathbb{I}_\varphi : [y,\infty[\,;[u_1,u_2] \text{ is defined}\\
&\Leftrightarrow \exists\,[u_1,u_2] \in \mathbb{I}_\varphi : [y,\infty[\text{ exists}\\
&\Leftrightarrow \mathsf{true}\ .
\end{aligned}$$

The corresponding statement for codomain is that $a^\urcorner = 0$ if and only if $\mathsf{inf}\,a = a$.

In this manner, all the presented theory of NL can be lifted to the infinite case. As in Chapter 3 one has to relax the algebraic structure. In the infinite setting INT forms a Boolean weak quantale. We mention only one more property for the infinite case; all the others are either straightforward or follow from the general laws for neighbourhoods in semirings presented in the next section.

Lemma 5.3.10 *In* NL *with infinite duration the semantics of the chop operator* $\frown$ *for an interval with infinite length* $([y,\infty[\models_{\mathcal{J},\mathcal{V}} \varphi\frown\psi)$ *is described by the formula* $\exists m : y \leq m < \infty \wedge [y,m] \models_{\mathcal{J},\mathcal{V}} \varphi \wedge [m,\infty[\models_{\mathcal{J},\mathcal{V}} \psi$. *For general intervals the chop is then expressed by*

$$\varphi\frown\psi \;\Leftrightarrow\; \exists x,y : \Big((\ell = x+y) \wedge (\boldsymbol{x} < \boldsymbol{\infty}) \wedge \Diamond_l\Diamond_r\big((\ell = x)\wedge\varphi\wedge\Diamond_r((\ell = y)\wedge\psi)\big)\Big) .$$

We already mentioned that NL subsumes logics like the one of Halpern and Shoham ([HS91]), the binary interval modalities of Venema ([Ven91]) and ITL ([HMM83]). Using the same arguments our extension now subsumes extensions of those logics for infinite intervals. In particular, it covers the logics presented in [Ska94, WX04, Rab00] and [ZVHX95].

5.4 Semiring Neighbours

Starting with the expressions for the neighbourhoods derived in the previous section and motivated by Theorem 5.3.7, we now give definitions that work in general lazy semirings with (co)domain. This paves the way for applying NL to hybrid systems. We simply replace sets of intervals by semiring elements, $\emptyset$ by 0 and inclusion $\subseteq$ by the natural order $\leq$. All proofs have been automated, the summarised results can be found in Appendix A. Hand-written proofs are given in [HM06, Höf07, HM08].

Definition 5.4.1 (semiring neighbours)
Assume a modal lazy semiring S.

1. a is a *left neighbour* of b (or $a \leq \diamondsuit_l b$ for short) iff $\lceil a \rceil \leq \lceil b$,
2. a is a *right neighbour* of b (or $a \leq \diamondsuit_r b$ for short) iff $\lceil a \leq \lceil b$,
3. a is a *left boundary* of b (or $a \leq \diamondsuit_l b$ for short) iff $\lceil a \rceil \leq \lceil b$,
4. a is a *right boundary* of b (or $a \leq \diamondsuit_r b$ for short) iff $\lceil a \leq \lceil b \rceil$.

We will see below that the use of $\leq$ is justified. Note that the b inside the diamond stands for boundary and is not related to the argument b, which is an arbitrary element. By *semiring neighbours* we mean both, left/right neighbours and boundaries.

Now we take a closer look at the definition and its interpretation in INT. It is straightforward to see the connection between semiring neighbours and the modalities of NL. As an example take the equivalence

$$i \models \Diamond_l \varphi \quad \text{iff} \quad \{i\} \leq \oslash_r \mathbb{I}_\varphi \ ,$$

for any interval i. The change in the direction is caused by different points of view. The original interpretation of $i \models \Diamond_l \varphi$ was that i *has* a left neighbour interval where φ holds. Our reading of $i \leq \oslash_r \mathbb{I}_\varphi$ is that i *is* a right neighbour of some interval in $\mathbb{I}_\varphi$. In our opinion the latter notation is more intuitive, since, looking at the figures of Section 5.3 $\Diamond_l \varphi$ is on the right hand side of the interval where φ holds.

Starting from the definitions of semiring neighbours we calculate an explicit form of these operators if the existence of a greatest element $\top$ is guaranteed. Such an element exists in nearly all semirings that occur in applications. In particular, all semirings which are built via a power set construction, like INT and PRO have a greatest element, namely the set of all elements. Moreover every omega algebra has a greatest element (cf. Lemma 3.4.4).

Lemma 5.4.2 *Assume a modal lazy semiring with greatest element $\top$. Neighbours and boundaries can be expressed as*

$$\oslash_l b = \top \cdot \lceil b \ , \qquad \oslash_r b = \lfloor b \cdot \top \ , \qquad \oslash_l b = \lceil b \cdot \top \qquad \textit{and} \qquad \oslash_r b = \top \cdot \lfloor b \ .$$

Consequently, $(\oslash_l b)\rceil = \lceil b$, $\lceil(\oslash_r b) = \lfloor b$, $\lceil(\oslash_l b) = \lceil b$ *and* $(\oslash_r b)\rceil = \lfloor b$.

As a direct consequence of the explicit expressions and the equation $\lceil(p \cdot \top) = p$, we have the following cancellation properties for nested neighbours

Corollary 5.4.3 (cancellation properties for nested neighbours) *The following laws can be used to simplify nested neighbours:*

$$\begin{array}{llll}
\oslash_l \oslash_r b = \oslash_r b & \textit{and} & \oslash_r \oslash_l b = \oslash_l b \ , \\
\oslash_l \oslash_r b = \oslash_r b & \textit{and} & \oslash_r \oslash_l b = \oslash_l b \ , \\
\oslash_l \oslash_l b = \oslash_l b & \textit{and} & \oslash_r \oslash_r b = \oslash_r b \ , \\
\oslash_l \oslash_l b = \oslash_l b & \textit{and} & \oslash_r \oslash_r b = \oslash_r b \ .
\end{array}$$

This corollary shows that Axiom 6 of [ZH98], which postulates that left and right neighbourhoods of an interval always end and start at the same point, is also a theorem in our setting.

To define boxes similar to $\Box_l$ and $\Box_r$ in the general setting we assume that the underlying semiring S is Boolean.

Definition 5.4.4 (perfect semiring neighbours)
Assume a modal lazy semiring S.

1. a is a *perfect left neighbour* of b (or $a \leq \boxed{n}_l b$) iff $a^{\rceil} \cdot \lceil \overline{b} \leq 0$,
2. a is a *perfect right neighbour* of b (or $a \leq \boxed{n}_r b$) iff $\overline{b}^{\rceil} \cdot \lceil a \leq 0$,
3. a is a *perfect left boundary* of b (or $a \leq \boxed{b}_l b$) iff $\lceil a \cdot \lceil \overline{b} \leq 0$,
4. a is a *perfect right boundary* of b (or $a \leq \boxed{b}_r b$) iff $a^{\rceil} \cdot \overline{b}^{\rceil} \leq 0$.

Parts (1) and (2) correspond to the box-operators of NL. By (3) and (4) we have an additional extension of NL. These two definitions provide "box operators" for the nested neighbourhood modalities $\Diamond_l \Diamond_r$ and $\Diamond_r \Diamond_l$, which are not defined in the semantics of NL in [ZH04].
To justify the definitions above we have

Lemma 5.4.5 *Each perfect neighbour (boundary) is a neighbour (boundary):*

$$\boxed{n}_l b \leq \diamondsuit\!\!\!\!\!\!\diamondsuit_l b \ , \qquad \boxed{n}_r b \leq \diamondsuit\!\!\!\!\!\!\diamondsuit_r b \ , \qquad \boxed{b}_l b \leq \diamondsuit\!\!\!\!\!\!\diamondsuit_l b \ , \qquad \boxed{b}_r b \leq \diamondsuit\!\!\!\!\!\!\diamondsuit_r b \ .$$

For boundaries this corresponds to the fact in modal logic that $\Box\phi \rightarrow \Diamond\phi$ iff the underlying relation is total, since every interval is a boundary of itself.

Corollary 5.4.6 *From the definitions we immediately get the box exchange rule*

$$a \leq \boxed{n}_l b \Leftrightarrow \overline{b} \leq \boxed{n}_r \overline{a} \ .$$

Like neighbours/boundaries we can characterise the box operators in an explicit form.

Lemma 5.4.7 *Perfect neighbours and boundaries can be expressed as*

$$\boxed{n}_l b = \top \cdot \neg \lceil \overline{b} \ , \quad \boxed{n}_r b = \neg \overline{b}^{\rceil} \cdot \top \ , \quad \boxed{b}_l b = \neg \lceil \overline{b} \cdot \top \ , \quad \boxed{b}_r b = \top \cdot \neg \overline{b}^{\rceil} \ .$$

Consequently, $(\boxed{n}_l b)^{\rceil} = \neg \lceil \overline{b}$, $\lceil (\boxed{n}_r b) = \neg \overline{b}^{\rceil}$, $\lceil (\boxed{b}_l b) = \neg \lceil \overline{b}$ *and* $(\boxed{b}_r b)^{\rceil} = \neg \overline{b}^{\rceil}$.

In the remainder of this section we show some properties of (perfect) neighbours and boundaries and compare them to properties of NL. To reduce calculations we introduce $\diamondsuit$ and $\boxdot$ as parameterised versions that can be instantiated by either $\diamondsuit\!\!\!\!\!\!\diamondsuit_l$, $\diamondsuit\!\!\!\!\!\!\diamondsuit_r$, $\diamondsuit\!\!\!\!\!\!\diamondsuit_l$ or $\diamondsuit\!\!\!\!\!\!\diamondsuit_r$ and $\boxed{n}_l$, $\boxed{n}_r$, $\boxed{b}_l$ or $\boxed{b}_r$, respectively. The instantiation must be consistent for all occurrences of $\diamondsuit$ and $\boxdot$. The following proofs are only done for one instance of $\diamondsuit$ or $\boxdot$; for all other instances they are similar. If the "direction" of $\diamondsuit$ or $\boxdot$ is important we use formulae like $\diamondsuit_l$ and $\boxdot_r$ where only one degree of freedom remains.

The above explicit forms (Lemma 5.4.2 and 5.4.7) show immediately that boxes and diamonds are connected via the de Morgan dualities, i.e., $\overline{\diamond \overline{a}} = \boxdot a$ and $\overline{\boxdot \overline{a}} = \diamond a$, if the underlying Boolean semiring is full; hence they form proper modal operators. In the lazy setting we get due to Corollary 3.5.4 a slightly different result.

Lemma 5.4.8 *Assume a Boolean modal lazy semiring.*

1. $\overline{\langle n \rangle_r \overline{b}} = [n]_r b \quad$ *and* $\quad \overline{[n]_r \overline{b}} = \langle n \rangle_r b$,
2. $\overline{\langle b \rangle_l \overline{b}} = [b]_l b \quad$ *and* $\quad \overline{[b]_l \overline{b}} = \langle b \rangle_l b$,

If the underlying semiring is weak, then the following inequations also hold.

3. $\overline{\langle n \rangle_l \overline{b}} \leq [n]_l b \quad$ *and* $\quad \overline{[n]_l \overline{b}} \leq \langle n \rangle_l b$,
4. $\overline{\langle b \rangle_r \overline{b}} \leq [b]_r b \quad$ *and* $\quad \overline{[b]_r \overline{b}} \leq \langle b \rangle_r b$.

The converse inequations of Part (3) and (4) do not hold, since in most cases $\top \not\leq \mathsf{F}$ (if there is at least one infinite element $\neq 0$). Hence $\overline{\langle n \rangle_l \overline{\top}} = \overline{\top \cdot \lceil 0} = \overline{\top \cdot 0} = \overline{\mathsf{N}} = \mathsf{F}$ and $[n]_l \top = \top \cdot \neg \lceil 0 = \top$. In particular, the inequations can be strengthend to equations if b is purely finite. Additionally, we show that some diamonds and boxes are lower and upper adjoints of Galois connections:

Theorem 5.4.9 *Diamonds and boxes form the following Galois connections.*

$$\langle n \rangle_r a \leq b \Leftrightarrow a \leq [n]_l b \quad \text{and} \quad \langle b \rangle_l a \leq b \Leftrightarrow a \leq [b]_l b \ .$$

Since Galois connections are useful as theorem generators and dualities as theorem transformers (e.g. [Bac02]) we get many properties of (perfect) neighbours and (perfect) boundaries for free and can simplify NL even more. For example we get directly by the Galois connection

Corollary 5.4.10

1. $\langle n \rangle_r$, $\langle b \rangle_l$, $[n]_l$ *and* $[b]_l$ *are isotone.*
2. $\langle n \rangle_r$, $\langle b \rangle_l$ *are disjunctive and* $[n]_l$, $[b]_l$ *are conjunctive.*
3. *The cancellation laws* $\langle n \rangle_r [n]_l a \leq a \leq [n]_l \langle n \rangle_r a$ *and* $\langle b \rangle_l [b]_l a \leq a \leq [b]_l \langle b \rangle_l a$ *hold.*

All these properties are standard implications of the Galois connection (compare e.g. Lemma 7.26 and Proposition 7.31 of [DP02]). Therefore no proofs are needed and the corresponding properties for NL come for free. Similar connections are not valid for left neighbours and right boundaries, but some implications can still be proved.

Theorem 5.4.11 *Assume again a Boolean modal weak semiring.*

$$\diamond\!\!\!\!\cdot_l a \leq b \Rightarrow a \leq \boxed{n}_r b \;, \qquad \diamond\!\!\!\!\cdot_r a \leq b \Rightarrow a \leq \boxed{b}_r b \;.$$

By lack of Galois connections, we do not have a full analogue to Corollary 5.4.10. Hence the following statements are not direct consequences.

Lemma 5.4.12 *Assume a Boolean lazy semiring.*

1. $\diamond\!\!\!\!\cdot_l$, $\diamond\!\!\!\!\cdot_r$, $\boxed{n}_r$ *and* $\boxed{b}_r$ *are isotone.*
2. *If S is right-distributive, then $\diamond\!\!\!\!\cdot_l$, $\diamond\!\!\!\!\cdot_r$ are disjunctive and $\boxed{n}_r$, $\boxed{b}_r$ are conjunctive.*

Since 0 is the least element with respect to $\leq$ and domain as well as codomain are strict, 0 is a neighbour and boundary of each element. Furthermore, special neighbours and boundaries are summarised in

Lemma 5.4.13 *The following properties hold in arbitrary Boolean modal lazy semirings.*

1. $\diamond 1 = \diamond \top = \boxdot \top = \top$.
2. $\diamond\!\!\!\!\cdot_r 0 = \diamond\!\!\!\!\cdot_l 0 = \boxed{n}_r 0 = \boxed{b}_l 0 = 0$ *and* $\diamond\!\!\!\!\cdot_l 0 = \diamond\!\!\!\!\cdot_r 0 = \boxed{n}_l 0 = \boxed{b}_r 0 = \mathsf{N}$.
3. *By isotony, $\ulcorner a \leq \diamond_l a$ and $a\urcorner \leq \diamond_r a$. Additionally, a is a left (right) boundary of itself, i.e., $a \leq \diamond\!\!\!\!\cdot_l a$ and $a \leq \diamond\!\!\!\!\cdot_r a$.*

Parts (1) and (2) show that all box operators satisfy the modal axiom M if the underlying semiring is full. Part (3) cannot be transferred from $\diamond$ to $\boxdot$, i.e., $x \leq \boxed{b} x$, $\ulcorner x \leq \boxed{n}_l x, \ldots$ do not hold, since in general $\ulcorner \overline{a} \neq \neg \ulcorner a$. Most proofs use the explicit forms for lazy semiring neighbours or the Galois connections. More properties can for example found in [HM08]

Since lazy semrings reflect aspects of infinity, we get some useful properties. Some are summarised in the following lemma.

Lemma 5.4.14 *Assume a Boolean modal lazy semiring.*

1. $\diamond\!\!\!\!\cdot_l \mathsf{F} = \diamond\!\!\!\!\cdot_r \mathsf{F} = \diamond\!\!\!\!\cdot_l \mathsf{F} = \diamond\!\!\!\!\cdot_r \mathsf{F} = \top$.
2. $b \leq \mathsf{N} \Leftrightarrow \diamond\!\!\!\!\cdot_r b \leq 0 \Leftrightarrow \diamond\!\!\!\!\cdot_r b \leq \mathsf{N}$.
3. $\boxed{n}_l \mathsf{N} = \boxed{b}_r \mathsf{N} = \mathsf{N}$ *and* $\boxed{n}_r \mathsf{N} = \boxed{b}_l \mathsf{N} = 0$.
4. $\overline{b} \leq \mathsf{N} \Leftrightarrow \mathsf{F} \leq b \Leftrightarrow \boxed{n}_r b = \top \Leftrightarrow \boxed{b}_r b = \top$.

The connection between (perfect) semiring neighbours and neighbourhood logic can be summarised as follows. Nearly all theorems of NL given in [RZ97, ZH98, ZH04] hold in the generalisation. Most of them already follow from the Galois connections and the lemmas presented before. The few which cannot be proved in the generalised setting need special properties of the time domain. An example is the density of $\mathbb{R}$. This property implies that each proper interval $[x, y]$ where $x \neq y$ can be split into two proper subintervals. A translation table between theorems presented in [ZH04] and our approach is given in Appendix A of [Höf05b].

With Corollary 5.4.3 we have already given cancellation laws for semiring neighbours. Using the explicit forms, we can show many more cancellation laws like

$$\diamondsuit_l \boxdot_l b = \boxdot_l b \quad \text{and} \quad \boxdot_l \diamondsuit_l b = \diamondsuit_l b \ . \tag{5.2}$$

In fact there are altogether 32 such laws, which are summarised in [Höf05b].

Within the calculations the combination $\boxdot \boxdot b = \overline{\diamondsuit\diamondsuit\overline{b}}$ holds in full semirings and turns out to be very useful. Furthermore, the "inner" operator dominates the "outer" one; i.e., in those cases, where $\boxdot\diamondsuit$ or $\diamondsuit\boxdot$ fulfils one of the cancellation laws, the expression is the same as $\diamondsuit\diamondsuit$ and $\boxdot\boxdot$, respectively. Further simplifications and properties are discussed in [Höf05b].

As stated in Lemma 5.3.10 $\phi^\frown\psi$ holds on an interval if and only if there is a finite time point m such that ϕ holds till m and ψ holds afterwards. We also showed the connection to NL. In the setting of the lazy semiring INT this becomes for arbitrary intervals i, j, k

$$\begin{aligned} i \models \phi^\frown\psi &\Leftrightarrow \exists\, j,k : i = j\,;k \wedge j \in \mathsf{fin}\,\mathbb{I}_\phi \wedge k \in \mathbb{I}_\psi \\ &\Leftrightarrow i \in (\mathsf{fin}\,\mathbb{I}_\phi)\,;\mathbb{I}_\psi \ , \end{aligned}$$

Hence, for $A \subseteq \mathbb{I}$, $A \models \phi^\frown\psi \Leftrightarrow A \subseteq (\mathsf{fin}\,\mathbb{I}_\phi)\,;\mathbb{I}_\psi$, so that in a general semiring we can simply identify $a^\frown b$ with $(\mathsf{fin}\,a)\cdot b$. This interpretation of chop is much easier than the ones of Theorem 5.3.5 and Lemma 5.3.10 and coincides well with the standard definitions in semirings and PRO (cf. Page 64). All the explicit treatment of the interval lengths can be skipped, since these are encoded in the concatenation of intervals, abstractly in the equation $a^\frown b = (\mathsf{fin}\,a)\cdot b$. This is a first step towards an algebraic combination of hybrid system analysis and logics.

In the previous section we derived compact closed semantics for CTL^*. We now show how neighbourhood concepts figure in the semantics of CTL^*.

By Theorem 5.2.5 we see that $[\![\mathsf{E}\varphi]\!]$ corresponds to a left boundary and $[\![\mathsf{A}\varphi]\!]$ to a perfect left boundary, i.e.,

$$[\![\mathsf{E}\varphi]\!] = \diamondsuit_l[\![\varphi]\!] \quad \text{and} \quad [\![\mathsf{A}\varphi]\!] = \boxdot_l[\![\varphi]\!] \ .$$

As far as we know, we are the first who built a bridge between CTL^* and NL. By this we can transfer knowledge between both logics. For example, from the cancellation laws for (perfect) semiring neighbours (Corollary 5.4.3 and Equations (5.2)), we obtain immediately

$$[\![\mathsf{EE}\varphi]\!] = [\![\mathsf{E}\varphi]\!] , \quad [\![\mathsf{AA}\varphi]\!] = [\![\mathsf{A}\varphi]\!] , \quad [\![\mathsf{EA}\varphi]\!] = [\![\mathsf{A}\varphi]\!] , \quad [\![\mathsf{AE}\varphi]\!] = [\![\mathsf{E}\varphi]\!] .$$

Of course, there are many more dual lemmas which we do not discuss here. The other two boundaries as well as all variants of (perfect) neighbours do not occur in CTL^* itself. However, the extension PCTL^* (e.g. [Rey01, Rey05]) of CTL^* provides operators for describing behaviour in the past. Therefore right boundaries occur in that setting.

5.5 Applying Algebraic Semantics to Hybrid Systems

In this section we sketch how to use the derived algebraic semantics of CTL^* and NL for hybrid system analysis. Since both, the algebra of hybrid systems and the algebraisation of these logics, are based on the same framework, cross-reasoning is possible and there is nearly nothing to do to apply the logics to hybrid systems.

We begin by combining computation tree logic with the algebra of hybrid systems. After that we will build a bridge between semiring neighbours and hybrid systems.

A next-time operator satisfying Equation (5.1) is not meaningful in continuous time models. Nevertheless we will give a short discussion in which circumstances a next-time operator is useful. For a discrete infinite set of durations D, e.g. $D = \mathbb{N}$, trajectories are isomorphic to nonempty finite or infinite words over the value set V. Moreover if V consists of values of computations, then the elements of PRO can be viewed as sets of computation streams (e.g. [BS01]). In these cases, a next time is meaningful. For example if $D = \mathbb{N}$, n can be chosen as the set of trajectories of length 1, i.e., as trajectories defined over the discrete interval $[0, 1]$.

Next we turn to the other operators of CTL^*. Since PRO is a Boolean modal lazy quantale, we simply reuse the above semantic equations (except those for X) and obtain semantics of a fragment of CTL^* for hybrid systems. The operators F, G and U can be realised as

$$[\![\mathsf{F}\varphi]\!] =_{df} \mathsf{F} \cdot [\![\varphi]\!] , \quad [\![\mathsf{G}\varphi]\!] =_{df} \overline{[\![\mathsf{F}\neg\varphi]\!]} , \quad [\![\varphi \,\mathsf{U}\, \psi]\!] =_{df} (\mathsf{fin}\, [\![\mathsf{G}\varphi]\!]) \cdot [\![\psi]\!] .$$

Note that in the first of these equations the F on the left is the CTL^* operator "finally", while on the right it is the largest finite element. A straightforward calculation shows that $\mathsf{F}\varphi = \mathsf{true}\,\mathsf{U}\,\varphi$ is still valid.

In fact we have already used similar formulas. By Theorem 5.2.5, we get

$$[\![\mathsf{F}p]\!] \;=\; \mathsf{F}\cdot[\![p]\!] \;=\; \mathsf{F}\cdot p\cdot\top = \diamondsuit p\;.$$

Similarly, we get $[\![\mathsf{G}p]\!] = \Box p$. Hence applications for CTL^* in the setting of hybrid systems are already given by Examples 4.5.5 and 4.5.14. Let us have a brief look at the former example.

Example 5.5.1 Again we return to the temperature control. Previously, we used the condition $A^{\mathrm{OFF}}\cdot A^{\mathrm{ON}} \leq \Box R_{[18,22]}$ to ensure that the temperature does not leave the given range. The logical counterpart would be that the system has to satisfy

$$\mathsf{G}\{18 \leq x \leq 20\}\;.$$

□

The standard construction for infinite iteration within the logic CTL^* is $\mathsf{GF}\varphi$. (" $\mathsf{F}\varphi$ has to hold on the entire subsequent path"). The algebraic equivalent is the modified diamond of Section 4 (cf. Definition 4.5.27) combined with iteration. This shows that CTL^* (using its algebraic semantics) is applicable to hybrid system analysis.

But a uniform algebraic base offers even more. We can do it the other way around and apply knowledge from hybrid system analysis to CTL^*. By definition of CTL^* we have $[\![p\mathsf{U}(\mathsf{G}p)]\!] = [\![\mathsf{G}p]\!]$. The above characterisation of U implies that the underlying semiring is safety-closed:

$$\Box p \;=\; [\![\mathsf{G}p]\!] \;=\; [\![p\mathsf{U}(\mathsf{G}p)]\!] \;=\; (\mathsf{fin}\,[\![\mathsf{G}p]\!])\cdot[\![\mathsf{G}p]\!] \;=\; \mathbb{F}p\cdot\Box p\;.$$

By $\mathsf{fin}/\mathsf{inf}$-splitting this is equivalent to $\Box p = \Box p\cdot\Box p$. The more general equation $[\![\mathsf{G}\varphi]\!] = [\![\mathsf{G}\varphi]\!]\cdot[\![\mathsf{G}\varphi]\!]$ does not hold. We can use the above correspondence between logical expressions and algebra and apply all lemmas and theorems of the previous chapters. For example Lemma 4.5.13(4) implies immediately the following logical expression.

$$[\![\mathsf{F}p \wedge (\varphi\mathsf{W}\psi)]\!] \;=\; [\![((\mathsf{F}p)\wedge\varphi)\mathsf{W}\psi \;\vee\; \varphi\mathsf{U}((\mathsf{F}p)\wedge\psi)]\!]\;,$$

where $\varphi\mathsf{W}\psi$ is weak until, defined as $\mathsf{G}\varphi \vee \varphi\mathsf{U}\psi$. That means that the difference with U is that there is no guarantee that a ψ will ever be satisfied. A direct proof using standard temporal logic is not trivial, whereas we were able to use Prover9 to automatically verify this property.

To conclude this chapter we now establish connections between neighbourhood logic and hybrid system analysis from an algebraic point of view. Again this is quite easy and there is nearly nothing to do. As mentioned before, CTL^* and NL bear strong similarities, since for example $[\![\mathsf{E}\phi]\!]$ corresponds to a left boundary. Hence we see that all discussed formalisms (hybrid systems,. CTL^* and NL) are closely related.

In PRO the left/right neighbours describe a kind of composability, i.e., for processes A and B,

$$A \leq \diamondsuit_l B \quad \text{iff} \quad \forall\, \tau \in A : \exists\, \tau' \in B : \tau \cdot \tau' \text{ is defined} , \tag{5.3}$$

$$A \leq \diamondsuit_r B \quad \text{iff} \quad \forall\, \tau \in A : \exists\, \tau' \in \mathsf{fin}\,(B) : \tau' \cdot \tau \text{ is defined} . \tag{5.4}$$

Both operators guarantee the existence of a composable element. $\diamondsuit_r B = B^{\urcorner} \cdot \top \neq 0$ guarantees that for every finite trajectory $\tau' \in \mathsf{fin}\, B$ there exists a trajectory τ that can continue τ'. If τ' in infinite, then there is no need to find a composable trajectory. Therefore $\diamondsuit_r B$ is a form of *liveness assertion*. In particular, the process $\diamondsuit_r B$ contains all trajectories that are composable with the "currently running" one. Note that in (5.3) the composition $\tau \cdot \tau'$ is defined if either $f(d_1) = g(0)$ (assuming $\tau = (d_1, f)$ and $\tau' = (d_2, g)$) or τ has infinite duration, i.e., $d = \infty$.

Looking at neighbours in the setting of hybrid systems and at the modal operators of Section 4.6 (Equations (4.9) and (4.10)) we see that neighbours are just a special case. However, their definition is useful, since they are defined on an arbitrary lazy semiring. (In contrast to the modal operators where a Boolean algebra is assumed.)

Lemma 5.5.2 *Assume a Boolean lazy quantale, then* $\diamondsuit_l b = \mathsf{E}b\,.\,\top$.

Proof. By straightforward calculations we get $\top \lfloor b = \top \lfloor {}^{\urcorner} b = \top \cdot {}^{\urcorner} b$ (see e.g., [Möl05b]). By indirect equality using the above equation and Lemma (5.4.2) we immediately get

$$a \leq \mathsf{E}b\,.\,\top \;\Leftrightarrow\; a \leq \top \lfloor b \;\Leftrightarrow\; a \leq \top \cdot {}^{\urcorner} b \;\Leftrightarrow\; a \leq \diamondsuit_l b\ .$$

□

From the definition of $\mathsf{A}b\,.\,w =_{df} \overline{\mathsf{E}b\,.\,\overline{w}}$ and Lemma 5.4.8(3), we can derive a dual statement for the universal continuation: $\mathsf{A}b\,.\,0 \leq \boxdot_l b$. As stated in the previous section the inequation can only be strengthend to an equation if n is purely finite.

In some situations one can also define left detachment. In this case $b \rfloor \top$ corresponds to $\diamondsuit_r b$. By the above connection we have again already given examples for the occurrence of neighbours in hybrid system analysis.

The situation for right/left perfect neighbours is more complicated. In [Höf05b] it is shown that $\boxdot_r B$ is the set of those trajectories that can be reached only from B, not from $\overline{B}$. Hence it describes a situation of guaranteed non-reachability from $\overline{B}$. The situation with $\boxdot_l$ is similar for finite processes, because of the symmetry between left and right perfect neighbours.

Chapter 6

Case Studies

Obodo n'ezu ezu azu nwa.
(Igbo and Yoruba (Nigeria) Proverb)

Throughout the thesis, we have illustrated the theory by simple examples, like the room heating or the bouncing ball. To round off the picture, this chapter presents case studies and proof experiments to show how concrete properties of hybrid systems, like safety and liveness, can be algebraically characterised and how off-the-shelf automated theorem provers can be used to verify them.

First, we revisit the railway constructions (cf. Section 2.5). By this example we show how hybrid systems can be modelled algebraically. The next case study describes a route planning problem. We show how to verify a safety properly if a specification is given. The last case study covers aspects of liveness of an assembly line scheduler.

6.1 Algebraic Railway Constructions

We briefly revisit the railway examples from Chapter 4. This example illustrates how a hybrid system, given by hybrid automata, can be expressed algebraically. It covers only modelling aspects and does not cover verification tasks; such exercises are presented in the other examples.

The first of the rail way examples was a gate controller. It models a circular track between 2000 and 5000 metres long that is equipped with a railway crossing with a gate; the architecture is given in Figure 2.9.

Before modelling the whole system, we model the train which was given by the hybrid

automaton in Figure 2.10. To derive the corresponding algebraic expression for this automaton, we follow the schema of Section 4.2. Since all control modes have the same structure, we define the following general processes:

$$\begin{aligned} T^{[v_1,v_2]} &=_{df} \{(d,x) \mid d \in \mathbb{R}_{\geq 0},\, v_1 \leq \dot{x} \leq v_2\} \,, \\ P_{dist} &=_{df} \{\underline{dist}\} = \{(0,x) \mid x = dist\} \,, \\ P_{\leq dist} &=_{df} \{\underline{dist}\} = \{(0,x) \mid x \leq dist\} \,. \end{aligned}$$

Process $T^{[v_1,v_2]}$ restricts the speed of the train to a velocity between v_1 and v_2; the duration of the trajectories is not restricted at all. In mode FAR for example the speed of the train is between 40 or 50 metres per second. Hence the mode is expressed by the process $T^{[-50,-40]}$. The zero-duration process P_{dist} is used to test whether the train is at a certain distance of $dist$ from the gate or not. For example P_0 tests if the train passes the gate at the moment.

When the *exit* event occurs the distance to the next gate has to be set. This was modelled by a jump condition in Figure 2.10. To model this condition algebraically, we use the compatibility relation $\asymp =_{df} \{(-100,x) \mid x \in [1900,4900]\}$. Depending on the length of the track it sets the distance after the train has passed the gate.

Using these elements the following algebraic expression for the train automaton results from our schema:

$$\mathrm{TR} =_{df} P_{\leq 5000} \cdot \Big((T^{[-50,-40]} \cdot P_{1000} \cdot T^{[-50,-30]} \cdot P_0 \cdot T^{[-50,-30]} \cdot P_{-100}) \asymp \Big)^{\circ} \,,$$

where $\circ$ is an iteration operator to be chosen (star, omega or dagger). The initial test $P_{\leq 5000}$ sets the starting point of the train: the distance between the gate and the train has to be smaller than $5000\,m$. As described in Section 4.2 the compatibility relation is employed at the right end of the repeated process. It is only needed at the point where we want to *enforce* a jump in the function describing the distance between the train and the gate. The other multiplications require the identity relation as compatibility relation, since we want to avoid jumps. Hence we do not need explicit compatibility relations for the other products. Note that in the algebraic expression we can replace omega by dagger, since the tests P_{dist} together with the given velocities of the train enforce that there are no Zeno-effects.

As the second component of the railroad gate control we have a gate automaton (Figure 2.11). We recapitulate the setting: The variable y represents the position of the gate in degrees; the gate is initially open; when a *lower* event is received, the gate starts closing and when a *raise* event is received, the gate starts opening. The given schema to convert hybrid automata to algebraic expressions yields

$$\mathrm{GA} =_{df} O \cdot \Big((M_l \cdot M_r)^* \cdot (C + O) \Big)^{\dagger} \,,$$

where

$$
\begin{array}{lll}
O &=_{df} \{(d, \mathsf{const}(90)) \mid d \in \mathbb{R}_{\geq 0}\} & \text{models control mode OPENED ,} \\
C &=_{df} \{(d, \mathsf{const}(0)) \mid d \in \mathbb{R}_{\geq 0}\} & \text{models control mode CLOSED ,} \\
M_l &=_{df} \{(d, y) \mid \dot{y} = -9,\, d \in \mathbb{R}_{\geq 0}\} & \text{models control mode DOWN ,} \\
M_r &=_{df} \{(d, y) \mid \dot{y} = 9,\, d \in \mathbb{R}_{\geq 0}\} & \text{models control mode UP ,}
\end{array}
$$

and const is the constant function (cf. Section 4.3). $M_l \cdot M_r$ is iterated because the gate can start opening even if it is not totally closed ($y = 0$) and it can start closing even if the gate is not absolutely opened ($y = 90$). Properly speaking the term for the gate controller is more complex when following the construction. It can be easily simplified by the regular identities given in Chapter 3 and basic facts about the processes modelling the control modes (e.g. transitivity).

The simplest way to combine both expressions is TR $\shuffle$ GA, where $\shuffle$ is the pointwise lifted parallel composition of Page 75. This algebraic expression contains all combinations of the train trajectories and the gate trajectories, e.g., the gate can be opened when the train passes. Hence, as discussed before, a simple combination is not useful.

To combine these two automata and to guarantee safety, we used a third automaton — a controller automaton — (cf. Figure 2.12). To simplify matters, we assume a reaction time of 0 seconds for the controller automaton. That means if an *approach* event is received, the controller immediately issues a *lower* event and when an *exit* event is received, the controller starts immediately an *raise* event; hence the gate starts closing. (Different delay times are also possible, altough the algebraic expressions become more complicated, the algebraic formulas would be similar.)

Using again the operator for parallel composition of Section 4 the composed automaton can be characterised by the following algebraic expression:

$$
\begin{aligned}
\mathrm{TG} \;=\; & \big(O \shuffle (P_{\leq 5000} \cdot T^{[-50,-40]} \cdot P_{1000})\big) \cdot \\
& \Big(\big((M_l \cdot C) \shuffle (T^{[-50,-30]} \cdot P_0)\big) \;\cdot\; \big(C \shuffle (T^{[-50,-30]} \cdot P_{-100})_{\asymp}\big) \cdot \\
& \big((M_r \cdot O) \shuffle (T^{[-50,-40]} \cdot P_{1000})\big)\Big)^{\dagger} .
\end{aligned}
$$

Let us have a closer look at the single components of composition. The first part $(O \shuffle (P_{\leq 5000} \cdot T^{[-50,-40]} \cdot P_{1000}))$ models the initial behaviour; the gate has to be open, the train starts somewhere before the gate (not farther than 5000 metres), and moves until it reaches the point $x = 1000$. Each of the components in the infinite iteration loop has as right operand of the parallel composition one control mode of the train automaton together with the attached event and as left operand the corresponding behaviour of the gate. Since the gate components end up in the modes C and O where the gate is opened or closed, respectively, processes like $M_l \cdot C$ can be lengthened to any duration longer than the shortest duration of M_l. Therefore we do not need a

constant function for the parallel composition (as discussed in Section 4.3). Note that the nested iteration of GA has been removed, because that behaviour cannot occur. Furthermore, this example might, in contrast to the algebraic expression of the train automaton, contain Zeno effects; therefore omega and dagger might behave differently and one has to choose dagger to be safe.

To conclude the example we discuss some aspects of safety for the gate controller. The algebra of processes not only compacts the description by a parallelised hybrid automaton, but also contains many aspects of safety. E.g., the expression $M_l \cdot C$ itself guarantees that the gate is closed at the time when the train passes the gate. This guarantee is not given in the original paper [Hen96]. Furthermore, it is easy to see that if the initial distance between the gate and the train is smaller than 1000, we have for the first factor of the above overall equation

$$(P_{<1000} \cdot T^{[-50,-40]} \cdot P_{1000}) \;=\; 0 \;.$$

Thus we know that such an initial distance is not safe, since it is not possible that the gate gets closed in time. This problem is not discussed in the original work [Hen96].

In general, if an algebraic expression or a part of it at a strict position (after a finite run) is equal to zero, the corresponding system is not safe. Another aspect of safety is the Zeno problem. In our example, Zeno effects can occur in the hybrid automaton as well as in our algebraic expressions. But those effects can be excluded by taking

$$\mathrm{TG} \sqcap \mathsf{N} \;,$$

as discussed in Section 4.4. Sometimes it is desirable and necessary to introduce range assertions. For instance, we may, besides the normal conditions of operation, want to guarantee that no train is faster than 40 metres per seconds (e.g. if there is construction work on the track). Then we have to modify the expression. Using the range assertions of Section 4.5 the algebraic expression can be modified to

$$\mathrm{TG} \sqcap \Box T^{[0,-40]} \;.$$

With this, we have a characterisation of the modified system and can now check safety, etc.

In a similar way one can build up an algebraic expression for the shared single-track railway example. Since it yields no new insights, it is left to the reader.

6.2 Route Planning

The following case study, adapted from [Höf08b], verifies a safety property for a given specification. A security service typically controls different locations. Due to safety

aspects the locations have to be checked periodically. But how can it be checked wether a given route between the locations guarantees that the places are checked in time? For simplicity, we assume that the security service only has to check three locations: a university, a disco and a bank. In Figure 6.1 a hybrid automaton is given that models all possible routes the security service can use when starting at university.

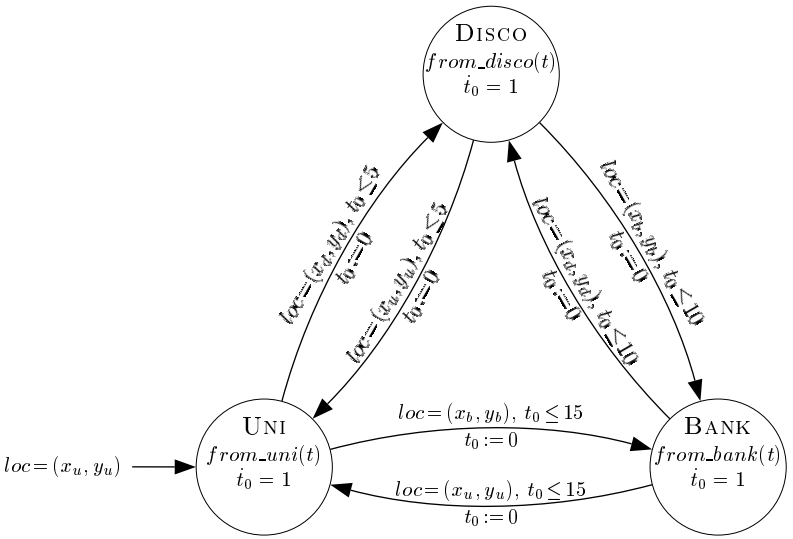


Figure 6.1: A simple system for route planning

We briefly explain the meaning of the automaton. Employees of the security service can be in three different states: either they are coming from the university (described by state UNI) or they are on their way from the BANK to a new control point or they have just controlled the DISCO. The functions $from\_uni$ describe the continuous behaviour of the hybrid system when moving from university (continuous behaviour in control mode UNI); they determine the specific location (x, y) at time t. Usually this function is specified by an initial value problem combined with (ordinary) differential equations. Special locations for university, bank and disco are denoted by (x_u, y_u), (x_b, y_b) and (x_d, y_d), respectively. If the bank or the disco is reached ($loc = (x_b, y_b)$ or $loc = (x_d, y_d)$), the automaton performs the respective transition. This state-changing situation represents the discrete part of the hybrid system. The other states and functions are built in a similar way. The clock t_0 measures the time when moving. To measure time between two locations a clock (the function t_0) is introduced. Time conditions like $t_0 \leq 5$, given at the edges, guarantee that the way between university

and disco takes at most 5 minutes; the way between disco and bank needs less than 10 minutes and the one between bank and university less than 15 minutes. After changing the state, the clock is reset to 0. Now we assume that the security service has to check every place at least every half an hour. Due to the small size it is easy to see that e.g. the circle starting at university and then via bank to disco and back to university satisfies the required safety condition, if it is repeated again and again. (Provided that there exist routes between the control points satisfying the time constraints.)

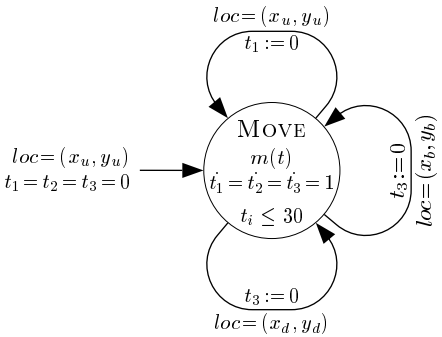


Figure 6.2: An alternative route planning automaton

To encode the time constraint that every location has to be visited every 30 minutes, one can use the hybrid automaton of Figure 6.2. The main idea is to have one state in which the service is moving. The action of moving is denoted by $m(t)$, e.g., $\dot{m}(t) = v$ if the movement is done with a constant velocity v, and the current position as initial condition $m(0) = (x_c, y_c)$.[10] Unfortunately, in this automaton the time constraints between the 3 locations cannot be encoded. To model the specification within hybrid automata one has to combine both automata presented. This yields an automaton with 4 clocks. To check the given safety property using one of these hybrid automata is not an easy and straightforward exercise.

But how can it be (automatically) checked that a run of a hybrid automaton satisfies a given specification, in general? We show that using the algebraic setting yields a surprisingly simple inequality for the above safety property that can easily be proved.

We determine an algebraic expression for the automaton of Figure 6.1. For that we define $V = \mathbb{R}^2$, where an element determines the current position (x, y). To construct

[10] This example is not realistic, but will illustrate the crucial ideas. In particular, typically the movement is not done with constant velocity, but is described by differential equations.

the processes corresponding to modes we use the simplified version and do not encode the mode names. For example

$$U =_{df} \{(d,g) \mid g(t) = from\_uni(t)\}$$

is the process for the mode UNI. The clock t_0 can be omitted since we have the duration d available and therefore the clock is redundant. Similar to U one can define processes for the modes DISCO and BANK. But, since the functions $from\_uni$, $from\_bank$ and $from\_disco$ are not specified we abstract to a general "move action". In particular, we define

$$A_n =_{df} \{(d,g) \mid d \leq n, g(t) = m(t)\} \ .$$

It describes all routes that the security service can use and take at most n minutes. To check if the security service is at a certain point, we use test (zero-duration trajectories):

$$\begin{aligned} AT_u &=_{df} \underline{(x_u, y_u)} = \{(0,g) \mid g(0) = (x_u, y_u)\} \ , \\ AT_b &=_{df} \underline{(x_b, y_b)} = \{(0,g) \mid g(0) = (x_b, y_b)\} \ , \\ AT_d &=_{df} \underline{(x_d, y_d)} = \{(0,g) \mid g(0) = (x_d, y_d)\} \ . \end{aligned}$$

These sets describe the situation when the security service is exactly at the locations university (AT_u), bank (AT_b) and disco (AT_d). In the remainder we use such elements to model tests and assertions. Now, we are able to describe the hybrid automaton of Figure 6.1 in an algebraic setting. The main construct is of the form $AT_u \cdot A_5 \cdot AT_d$ which describes all possible ways from university to the disco that take at most five minutes. Since we assume that the three locations under consideration are at different places the movement from one place to another needs some minimal amount of time. Therefore trajectories in structures like $AT_u \cdot A_5 \cdot AT_d$ cannot be infinitely short. Hence these structures avoid Zeno effects and we can use omega. The whole security system is described by

$$\begin{aligned} AT_u \cdot \big(& AT_u \cdot A_5 \cdot AT_d + AT_d \cdot A_5 \cdot AT_u + \\ & AT_d \cdot A_{10} \cdot AT_b + AT_b \cdot A_{10} \cdot AT_d + \\ & AT_b \cdot A_{15} \cdot AT_u + AT_u \cdot A_{15} \cdot AT_b \big)^{\omega} \ . \end{aligned}$$

Remember that we want to check that, for a given trajectory of the hybrid automaton, the security service checks every location at least every 30 minutes. Let us consider the following (infinite) route for the security service:

$$\tau =_{df} (AT_u \cdot A_5 \cdot AT_d \cdot A_{10} \cdot AT_b \cdot A_{15})^{\omega} \ .$$

It describes the situation where the security service starts at university, travels to the disco and drives on to the bank. After checking this locations it returns back to university and starts again its route. It is straightforward to show that τ is a trace of the hybrid automaton's encoding of Figure 6.1. To formulate the safety criterion for visiting each place at least once in 30 minutes, we have to check whether $\tau \leq (A_{30} \cdot AT_u)^\omega \sqcap (A_{30} \cdot AT_d)^\omega \sqcap (A_{30} \cdot AT_b)^\omega$ holds. By Boolean algebra it is equivalent that

$$\tau \leq (A_{30} \cdot AT_u)^\omega \ , \qquad \tau \leq (A_{30} \cdot AT_d)^\omega \qquad \text{and} \qquad \tau \leq (A_{30} \cdot AT_b)^\omega \ . \tag{6.1}$$

We only show that the second equation can easily checked by hand; the other inequalities can be shown similarly. In the next section we present a possibility to automate such calculations. By isotony and definition of A_n we get

$$AT_u \cdot A_5 \cdot AT_d \cdot A_{10} \cdot AT_b \cdot A_{15} \ \leq \ A_5 \cdot AT_d \cdot A_{10} \cdot A_{15} \ \leq \ A_5 \cdot AT_d \cdot A_{25} \ .$$

Hence it is sufficient to show that $(A_5 \cdot AT_d \cdot A_{25})^\omega \leq (A_{30} \cdot AT_d)^\omega$. By unfold (3.16), Lemma 3.4.5(1), isotony and unfold again, we get

$$\begin{aligned} (A_5 \cdot AT_d \cdot A_{25})^\omega &= (A_5 \cdot AT_d \cdot A_{25}) \cdot (A_5 \cdot AT_d \cdot A_{25})^\omega \\ &\leq A_5 \cdot AT_d \cdot (A_{25} \cdot A_5 \cdot AT_d)^\omega \\ &\leq A_{30} \cdot AT_d \cdot (A_{30} \cdot AT_d)^\omega \\ &= (A_{30} \cdot AT_d)^\omega \ . \end{aligned}$$

This calculation shows that the chosen trace satisfies the safety criterion. In the algebraic setting it is a simple and short calculation, whereas in the setting of hybrid automata it was not possible in a straightforward way.

We will now check the Equations (6.1) fully automatically using Prover9. Standard theorem provers are not able to handle simple arithmetic. Usually this is achieved through higher-order theorem proving at the expense of computational power and more user interaction. Unfortunately this is one of the main disadvantages of automated theorem provers at the moment and deserves more research (cf. Chapter 7). However, since we only need simple arithmetic, we can encode the relationship between different elements like $A_5 \cdot A_{15} \leq A_{30}$ by hand. Obviously it is not difficult to produce such formulas with an automated preprocessor. The three equations are encoded by

```
formulas(goals).
  all u all d all b(
      u;u=u & u<=1 & d;d=d & d<=1 & b;b=b & b<=1   %preconditions
    ->
      (u;a5;d;a10;b;a15)^ <= (a30;u)^ &
      (u;a5;d;a10;b;a15)^ <= (a30;d)^ &            %the 3 equations
      (u;a5;d;a10;b;a15)^ <= (a30;b)^).
end_of_list.
```

In the code `u` corresponds to AT_u, `d` to AT_d, `a5` to A_5, etc. Since AT_u, AT_d and AT_b are zero-duration processes and therefore tests, we have to specify tests for Prover9. This can be done in a general setting or by specifying properties of tests (cf. Section 3.5). The preconditions reflect the two main properties for tests, namely that tests are idempotent subidentities. Prover9 can prove each of the equations.

That shows that algebraic reasoning for hybrid systems is indeed feasible. In particular, we have presented a safety property for a concrete hybrid system and have proved it fully automatically. Therefore the algebraic approach provides an interesting new way of verifying hybrid systems. It is straightforward to extend the above example. For instance, one can add more locations or one can refine the safety property (e.g. "The security service has to drive to a petrol station every 10 hours and refuel there for 5 minutes".) All these extensions do not change the algebra and/or the way of verifying the specification.

Verifying larger systems might need more time to prove properties fully automatically. But, checking properties are usually done in advance and not in real time. Moreover Prover9 can prove even complex properties in reasonable time; see e.g. Back's atomicity refinement law in [HS08a]. Therefore we expect that one also can use our approach for larger systems.

6.3 An Assembly Line Scheduler

To further underpin our approach we sketch yet another case study that verifies a liveness criterion. It is adapted from an example given by Henzinger, Horowitz and Majumdar in [HHM99].

We consider an assembly line scheduler that has to assign elements from an incoming stream to one of two assembly lines. New parts occur every four minutes in the stream. The lines themselves process the parts at different speeds: jobs travel between one and two meters per minute on the first line, while on the second the speed is between two and three metres per minute. The first line is three metres, the second six metres long. Once the lines finish a job, they insert cleaning phases of two and three minutes,

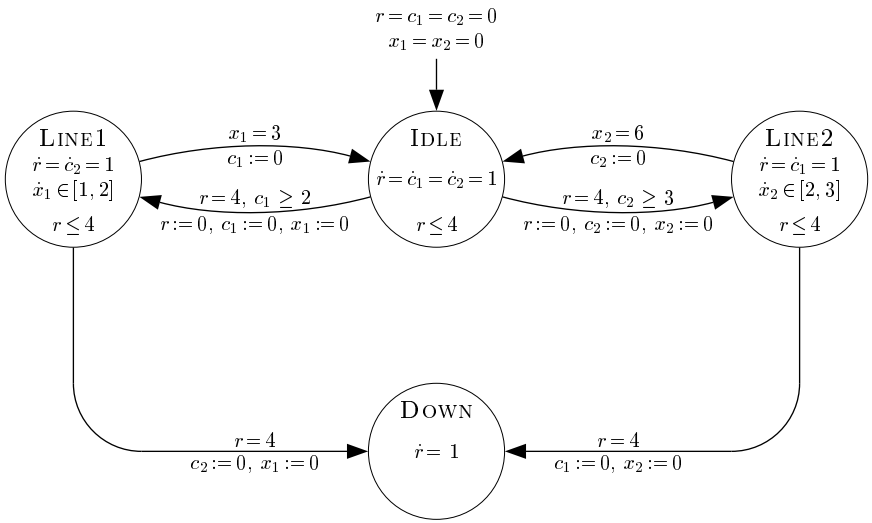


Figure 6.3: An assembly line scheduler

respectively, during which no job can be taken up. The whole system accepts a job if both lines are free, and at most one is cleaning up. If the system cannot accept a job it shuts down.

The system is modelled by the hybrid automaton given in Figure 6.3. There are four states: in IDLE no jobs are being processed; in LINE1 and LINE2 the lines for processing jobs are modelled; in DOWN the system shuts down. The real-valued variables x_1 and x_2 measure the distance a job has travelled along the first and second line, respectively. The real-valued variables c_1 and c_2 indicate the amount of time for cleaning up. Finally the variable $r \in \mathbb{R}$ measures the elapsed time since the last arrival of a job. In the hybrid automaton several differential equations are omitted. Following Section 2, conditions of constancy like $\dot{c}_1 = 0$ are left out to improve readability. That means that in mode IDLE the equations $\dot{x}_1 = 0$ and $\dot{x}_2 = 0$ also have to be satisfied.

As a liveness property one wants to prevent the system from going down. In [HM00] it is mentioned that any feasible schedule must choose the first line infinitely often. We will characterise and verify this liveness property in our algebraic setting. To derive an algebraic expression for the liveness criterion we define processes L_1', L_2', ID' and

D' for the nodes LINE1, LINE2, IDLE and DOWN respectively.

$$\begin{aligned}
L_1' &=_{df} \{(d,g') \mid \dot{r}=1,\ \dot{c}_2=1,\ \dot{c}_1=\dot{x}_2=0,\ \dot{x}_1\in[1,2]\}\ ,\\
L_2' &=_{df} \{(d,g') \mid \dot{r}=1,\ \dot{c}_1=1,\ \dot{c}_2=\dot{x}_1=0,\ \dot{x}_2\in[2,3]\}\ ,\\
I_D' &=_{df} \{(d,g') \mid \dot{r}=1,\ \dot{c}_1=\dot{c}_2=1,\ \dot{x}_1=\dot{x}_2=0\}\ ,\\
D' &=_{df} \{(d,g') \mid \dot{r}=1,\ \dot{c}_1=\dot{c}_2=\dot{x}_1=\dot{x}_2=0\}\ ,
\end{aligned}$$

where $d \in \mathbb{R}_{\geq 0}$ and g' is defined as $g' = r \times c_1 \times c_2 \times x_1 \times x_2$. The condition $r = 4$ is modelled by the test $R =_{df} \{(4,w,x,y,z) \mid w,x,y,z \in \mathbb{R}\}$. As we can see in Figure 6.3, the control modes LINE1, LINE2 and IDLE have to be left if r equals 4. Therefore the formulas $L_1' \cdot R$, $L_2' \cdot R$ and $I_D' \cdot R$ exclude from the above processes those trajectories violating the flow conditions and the jump conditions. So far we followed the schema of Section 4.2. In particular we used r. This variable models a global clock inside the hybrid automaton. However, there is no need to do so. In the algebraic setting we can omit the variable r; the duration allows us to argue about time constraints. From the given specification, like the length of the lines, we derive new processes as follows:

$$\begin{aligned}
L_1 &=_{df} \{(d,g) \mid \dot{c}_2=1,\ \dot{c}_1=\dot{x}_2=0,\ \dot{x}_1\in[1,2],\ 1.5\leq d\leq 3\}\ ,\\
L_2 &=_{df} \{(d,g) \mid \dot{c}_1=1,\ \dot{c}_2=\dot{x}_1=0,\ \dot{x}_2\in[1,2],\ 2\leq d\leq 3\}\ ,\\
I_D &=_{df} \{(d,g) \mid \dot{c}_1=\dot{c}_2=1,\ \dot{x}_1=\dot{x}_2=0,\ d\leq 4\}\ ,\\
D &=_{df} \{(d,g) \mid \dot{c}_1=\dot{c}_2=\dot{x}_1=\dot{x}_2=0\}\ ,
\end{aligned}$$

where this time g is defined as $g = c_1 \times c_2 \times x_1 \times x_2$. To further argue about time we define processes for time measurement, similarly to the one of the previous example. This again equips Prover9 with simple arithmetic.

$$A_n =_{df} \{(d,g) \mid d\leq n\}\ ,\ A_{=n} =_{df} \{(d,g) \mid d=n\} \text{ and } A_{<n} =_{df} \{(d,g) \mid d<n\}\ .$$

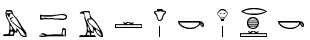
The former contains only those trajectories whose duration is at most n. The trajectories of the second process have exactly duration n; the third is their difference. By this $L_1 \leq A_3$, $L_2 \leq A_3$, $I_D \leq A_4$, $L_1 \sqcap A_{<1.5} = \emptyset$ and $L_2 \sqcap A_{<2} = \emptyset$. The last two formulas exclude undesirable trajectories with short durations from the processes. After setting the basics, we now return to the liveness criterion that any feasible schedule must choose the first line infinitely often. A feasible schedule is one that never leaves the modes LINE1, LINE2 and IDLE. The algebraic counterpart is $FS =_{df} ((I_D \cdot L_1 \cdot I_D + I_D \cdot L_2 \cdot I_D) \sqcap A_{=4})^\omega$. The part $\sqcap A_4$ guarantees that the corresponding hybrid automaton is in mode IDLE, where new parts occur every four minutes. By this the liveness requirement becomes

$$((I_D \cdot L_1 \cdot I_D + I_D \cdot L_2 \cdot I_D) \sqcap A_{=4})^\omega \leq (\mathsf{F} \cdot L_1)^\omega\ .$$

This statement can easily be automated; a proof by hand is fairly standard and easy for a reader with some experience in algebraic calculations. By this we have verified the liveness criterion of [HM00]. The criterion does not state anything about the existence of a feasible schedule. In the algebraic setting there is no feasible schedule if and only if FS does not contain a single trajectory, i.e., $\mathrm{FS} = \emptyset$. For a given schedule it can be checked easily if it is an element of FS. This can be shown similar to the previous case study. This example shows again that an algebraic approach yields simplifications in hybrid system analysis and opens a new way for automation using off-the-shelf automated theorem prover.

Chapter 7

Conclusion



Ptahhotep

The intention of this book was the investigation of hybrid systems under an algebraic view. Special attention was laid on a formalisation of verification tasks. The main contribution is therefore the development of an algebra of hybrid systems and the analysis of its structure and its properties. We also showed how to derive algebraic compact closed semantic expressions for logics. Both, the algebraic frameworks for these logics and the algebra of hybrid systems are based on the same algebraic structure and therefore cross-reasoning is possible.

In this closing chapter, we summarise the work and give some ideas for future research.

7.1 Summary

This book provides a comprehensive algebraic theory of hybrid systems based on lazy semirings and lazy iteration algebras. Full semirings are well known in computer science and emerge as fundamental structures in computing. These algebraic structures found widespread applications ranging from program analysis to concurrency control. Although one has to take some care, since the basic laws of lazy semirings are weaker than those for standard ones, things worked out reasonably well and many results come for free. We showed that hybrid systems are yet another application for this algebraic theory.

In Chapter 4 we have presented a model of trajectories and processes which then has been abstracted to admit a general semiring view. This model is able to cover hybrid system analysis. We have shown how to embed hybrid automata into that setting. Based on an analysis of the purely finite and purely infinite parts of behaviours we have demonstrated how Zeno effects can conveniently be handled. We have given algebraic definitions of several composition operators for hybrid systems. Moreover we have discussed safety and liveness properties as well as time restrictions and range assertions and certain temporal operators. It should be noted that nevertheless the whole development is based on few and well-known algebraic concepts.

Since the theory of semirings is completely first-order and Horn, it lends itself to mechanisation using off-the-shelf theorem provers such as Prover9. Hence the field of hybrid systems was tractable at an abstract level. In particular we used, whenever possible, theorem provers to verify and prove the theory. However, sometimes a combination between automatic theorem provers and domain specific solvers is desirable (cf. Chapter 6).

The algebra covers hybrid programs and differential dynamic logic in a straightforward way. Afterwards, we have given algebraic semantics for the branching time logic CTL^* and the neighbourhood logic NL. Since these logics and hybrid systems, are based on the same algebraic structure, the algebraisation allows immediately cross-reasoning. In particular, the application of CTL^* and NL to hybrid systems becomes quite easy. Since CTL^* subsumes its sublogics CTL and LTL and NL subsumes interval logics like ITL and the duration calculus, we have also derived an implicit algebraic characterisation for these logics.

The book has been supplemented by several examples and case studies.

7.2 Future Work

Although the book results in a coherent family of algebraic calculi of hybrid systems, some research questions are still open and need further investigation. We briefly present ideas for future research from different areas of computer science which we encountered during preparation of this book. In detail, we cover hybrid system analysis, algebraic questions, theorem proving tasks and links to game theory.

So far we applied the developed theory to small examples. Hence one should try to analyse a much greater case study. However, it is hard to get data for real examples.

In Chapter 4 we showed how hybrid systems can be handled algebraically. System properties such as non-deterministic choices, change over time, interaction and communication, iterative and limit behaviours can be modelled algebraically in an abstract, concise and uniform way for large model classes. Over the last few years, there

was a shift of focus within computing from qualitative to quantitative dynamic analysis. Instead of asking whether a hybrid system reaches some conditions or whether some properties hold while executing a loop, questions like "How high is the probability that a certain goal will be reached?" or "What will be the distribution of molecules in a cell after the following reactions have taken place?" now need to be answered.

Therefore, stochastic approaches are essential. Based on hybrid automata the notions of *stochastic hybrid automata* and *probabilistic hybrid systems* were introduced [CL06, APLS08, SAFM06]. An algebraic treatment for hybrid systems with respect to probabilistic aspects is missing. First steps towards an algebraic treatment of propabilistic systems are already done: McIver, Gonzalia, Cohen and Morgan used a variant of Kleene algebra for the verification of probabilistic programs and protocols [MGCM08]. First steps towards an algebraic view for probabilistic programs are done by Hayes, Meinicke and Solin [MH08, MS08]. But a number of important challenges need to be faced. The first challenge is the interaction between probabilistic and non-deterministic choice for various semantics ranging from relations to trajectories and processes. Convenient algebraic axioms and a generic way in which quantitative behaviour can be integrated into an algebraic framework need to be found and developed.

Adding probabilistic operators to an algebra is one task. Another one might be to change the iteration operators. One of the presented example was the bouncing ball. The loss of energy at each bounce makes the subsequent jumps closer and closer together in time. However the jumps have the same form. In our approach we handled this behaviour by introducing a damping factor. Such a construction is not always naturally given. An example is a swinging pendulum which constantly loses energy. Due to this the iteration operations are sometimes not flexible enough, since they repeat a given trajectory without modifying it. A possible solution is to combine the iteration operation with a function f that modifies the iterated element in each step. First steps for such an approach are presented in [Höf06] where the notion of *f-generated Kleene algebra* is introduced. At first glance this approach seems quite promising. Unfortunately, most of the regular identities, like $a^* \cdot a^* = a^*$, do not hold any longer. The behaviour of these new iteration operators deserves more research.

The last result to be discovered is an appropriate algebraic characterisation of the dagger operator. We have shown that omega does not reflect infinite behaviour in a proper way. Therefore we introduced the dagger operator for processes. Unfortunately, we do not know its position within the set of all fixpoints $x = a \cdot x$; at least we know that it is a fixpoint. It should be investigated if it can be expressed in simple first-order style axioms.

The experiments using Prover9 also pose some interesting research questions for automated deduction.

First, equational reasoning should be complemented by reasoning with inequalities (viz. chaining calculi), an issue that has so far rather been neglected in implementations. We have encoded inequalities as predicates in Prover9 together with the obvious axioms. We used this approach to show parts of Lemma 4.5.2, Lemma 4.5.13 and for the case studies. Moreover by this method we could automatically verify some key refinement laws for concurrent systems [HS07]. The equational coding failed on these examples. An encoding based on inequalities and in particular a theorem prover that can handle chaining rules for transitive relations might resolve this problem [BKS85, BG98].

Second, as shown in Chapter 6, verification tasks often need domain-specific knowledge. This is usually achieved through higher-order theorem proving at the expense of computational power or through model checking at the expense of expressive power. Hence concrete verification tasks can be automated algebraically up to domain specific calculations. Our example required simple arithmetics. Other proofs might require solvers, e.g., for data structures like lists, arrays or stacks or for more complex numeric domains. Integrating domain-specific solvers into state-of-the-art theorem provers would therefore have immediate practical relevance for analysis and verification programs and hybrid systems. A first step towards the integration of domain-specific solvers is SPASS+T [PW06]. This system is an extension of the superposition-based theorem prover SPASS [WSH$^+$07] that allows to enlarge the reasoning capabilities of SPASS using an arbitrary SMT procedure for arithmetic and free function symbols.

A last topic for future research was already touched in Section 4.6 where we sketched a connection with games. As Sintzoff [Sin04] has shown, the theory of games helps in understanding control systems as well as hybrid and reactive ones, since it deals with interaction between dynamics. Game theory and game algebra should help to obtain improved controllers for hybrid systems.

Appendix A

Automation in Lazy Kleene Algebras

Fata viam invenient.
Virgil (Aeneis X)

One advantage of an algebraic approach is the opportunity of automating proofs. A proof script does not only save time if it is fully automated, even if it needs some additional hypotheses as input it acts as a "certificate", increasing the confidence in the results. We used Prover9 to verify the algebraic parts of the thesis.

This appendix provides the necessary information and results concerning our proof experiments. First, we give templates for lazy semirings (Kleene and omega algebras) in formats for Prover9 and TPTP. This allows the interested reader to prove or disprove conjectures about the lazy structures presented. In the second part we summarise the results of the automated proofs of this thesis.

A.1 Templates

We provide proof templates for lazy omega algebras. Since we used Prover9 to verify the lemmas and theorems throughout the thesis, we first present a template for this paramodulation-based theorem prover. It allows to encode the axioms in an intuitive way, i.e., it accepts operators in infix, prefix and postfix notation; hence it is easy to use. Moreover a quantification of the variables involved is often not necessary.

Another advantage of Prover9 is that it is equipped with a tool for counterexample search, namely Mace4 [McC] which accepts the same input file.

```
% LANGUAGE SPECIFICATION
 op(500, infix,   "+" ).       % choice
 op(490, infix,   ";" ).       % composition
 op(700, infix,   "<=").       % order
 op(480, postfix, "*" ).       % finite iteration
 op(450, postfix, "^" ).       % infinite iteration (omega)

% AXIOMS
 formulas(sos).
   % additive, commutative and idempotent monoid
      x+(y+z) = (x+y)+z.
      x+0 = x.
      x+y = y+x.
      x+x = x.
   % multiplicative monoid
      x;(y;z) = (x;y);z.
      x;1 = x.
      1;x = x.
   % natural order
     x<=y <->  x+y=y.
   % distributivity and isotony
      (x+y);z = x;z+y;z.
      y<=z -> x;y<=x;z.
   % annihilation
      0;x=0.
   % finite iteration (star)
      1+x;x* = x*.
      (x;y+z)<=y -> x*;z<=y.
   % infinite iteration (omega)
      x;x^ = x^.
      y<=x;y+z -> y<=x^+x*;z.
  end_of_list.

 % CONJECTURE
  formulas(goals).
    %lemma to be proved
  end_of_list.
```

Next we present the encoding in TPTP-style. Sutcliffe's and Suttner's TPTP [SS98] (Thousands of Problems for Theorem Provers) is not only a library of test problems

for automated theorem proving (ATP) systems, but also provides the TPTP syntax, a uniform base for all established theorem provers. There are utilities to convert the problems to existing ATP systems' formats available. For more details see the technical manual [SS98].

Since the header section of a TPTP-file contains information about the problem we skip this part. We only give the formulas.

```
%----additive, commutative and idempotent monoid
 fof(additive_associativity,axiom,(
     ! [X,Y,Z] : addition(X,addition(Y,Z)) = addition(addition(X,Y),Z) )).
 fof(additive_identity,axiom,(
     ! [X] : addition(X,zero) = X )).
 fof(additive_commutativity,axiom,(
     ! [X,Y] : addition(X,Y) = addition(Y,X) )).
 fof(additive_idempotence,axiom,(
     ! [X] : addition(X,X) = X )).
%----multiplicative monoid
 fof(multiplicative_associativity,axiom,(
     ! [X,Y,Z] : multiplication(X,multiplication(Y,Z)) =
                 multiplication(multiplication(X,Y),Z) )).
 fof(multiplicative_right_identity,axiom,(
     ! [X] : multiplication(X,one) = X )).
 fof(multiplicative_left_identity,axiom,(
     ! [X] : multiplication(one,X) = X )).
%----natural order
 fof(order,axiom,(
     ! [X,Y] : ( leq(X,Y) <=> addition(X,Y) = Y ) )).
%----distributivity and isotony
 fof(left_distributivity,axiom,(
     ! [X,Y,Z] : multiplication(addition(X,Y),Z) =
                 addition(multiplication(X,Z),multiplication(Y,Z)) )).
 fof(right_isotony,axiom,(
     ! [X,Y,Z] : ( leq(Y,Z)
       => leq(multiplication(X,Y),multiplication(X,Z)) ) )).
%----annihilation
 fof(left_annihilation,axiom,(
     ! [X] : multiplication(zero,X) = zero )).
%----finite iteration (star)
 fof(star_unfold_right,axiom,(
     ! [X] : addition(one,multiplication(X,star(X))) = star(X) )).
 fof(star_induction_left,axiom,(
     ! [X,Y,Z] : ( leq(addition(multiplication(X,Y),Z),Y)
       => leq(multiplication(star(X),Z),Y) ) )).
```

```
%----infinite iteration (omega)
 fof(omega_unfold,axiom,(
     ! [X] : multiplication(X,omega(X)) = omega(X) )).
 fof(omega_co_induction,axiom,(
     ! [X,Y,Z] : ( leq(X,addition(multiplication(Y,X),Z))
       => leq(X,addition(omega(Y),multiplication(star(Y),Z))) ) )).
%----CONJECTURE
 fof(goals,conjecture,(
     %lemma to be proved
    )).
```

A.2 Automated Proofs

Not long ago, it was common belief that off-the-shelf automated provers cannot handle structures of the complexity used. Throughout the thesis project we have shown that Prover9 can handle algebraic structures. We now summarise the results of our proof experiments using Prover9. The version used is April-2007, other version might slow down or speed up the proof times.

In practice, basic properties can be proved with the standard axioms in a few seconds. In order to obtain ATP proofs of the more difficult properties described in this thesis, further lemmas had sometimes be added to the axioms. Useful additions could be reflexivity and transitivity of the order, isotony of all operations or the join splitting axiom. In the case where tests occur the proofs can be speed up by (llp), (gra), (lrp) or (wgra). Even if the proof needs some interaction and additional hypotheses, it is not useless since it can be seen as "certificate", increasing the confidence in the result. All additional lemmas had earlier been proved from the axioms.

We used a computer with a 3.0 GHz Intel Pentium 4 CPU, with 2 GB memory and hyper-threading, running a Linux 2.6 operating system.

The results and methods used for each law are shown in Table A.1. The first column refers to the lemma or theorem to be proved, the second recapitulates its main content without giving all assumptions. The next two columns indicate if some unnecessary axioms were removed from or additional hypotheses were added to the input: "yes" stands for some removal/addition and "—" indicates no removal/addition. The penultimate column shows wether a proof goal is split. Proofs are often faster if an equation $s = t$ is split into $s \leq t$ and $t \leq s$; equivalences are split into two implications. Finally, the last column indicates the CPU time in seconds taken for the successful proof. All input files can be found at a website [Höf]. From these one can generate the output files which include the proofs. If one does not trust automated systems one can check the proof step by step (another advantage of Prover9).

Table A.1: Summarised results using Prover9

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[12]	**t[s]**
Eq. (3.11)	one star unfold axiom can be skipped	—	—	yes	0
Eq. (3.16)	the omega unfold axiom can be relaxed	—	—	—	0
Lm. 3.2.2[14]	right-super-distributivity is equivalent to right isotony	—	—	—	127
Lm. 3.2.2[14]	right isotony implies right-super-distributivity	—	—	yes	0
Lm. 3.2.2[14]	right-super-distributvity implies right isotony	—	—	yes	0
Lm. 3.2.3	shunting rule for Boolean lazy semirings	—	—	—	424
Lm. 3.2.10	indirect equality	—	—	—	0
Lm. 3.4.1(1)	$a \leq a^*$	—	—	—	0
Lm. 3.4.1(2)	isotony of star	—	—	—	0
Lm. 3.4.1(3)	idempotence of star	—	—	—	4
Lm. 3.4.1(4)	$a^* \cdot a \leq a \cdot a^* = a^+$	—	—	—	0
Lm. 3.4.1(5)	$(a \cdot b)^* \leq (a + b)^*$	—	—	—	284
Lm. 3.4.2	$a \cdot a^* = a^* \cdot a$	—	—	—	396
Lm. 3.4.3(1)	isotony of omega	—	—	—	7
Lm. 3.4.3(2)	weak idempotence of omega	—	—	—	5
Lm. 3.4.3(3)	$a^* \cdot a^\omega = a^\omega$	—	—	—	45
Lm. 3.4.3(4)	$a \cdot a = a \Rightarrow a^\omega = a \cdot \top$	—	—	—	59
Lm. 3.4.3(5)	$c \leq a \cdot c \Rightarrow c \leq a^\omega$	—	—	—	1

[11] An equation $a = b$ as goal is split into the two inequations $a \leq b$ and $b \leq a$; similarly, an equivalence as goal is split into implications.

[12] This example demonstrates that splitting an equivalence (equation) often yields much faster proofs.

Table A.1: Summarised results using Prover9 (continued)

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[13]	**t[s]**
Lm. 3.4.4	greatest element exists and is left ideal	—	—	—	61
Lm. 3.4.5(1)	$a \cdot (b \cdot a)^\omega = (a \cdot b)^\omega$	—	—	yes	13
Lm. 3.4.5(1)	$a \cdot (b \cdot a)^\omega = (a \cdot b)^\omega$	—	yes	yes	0
Lm. 3.4.5(2)	$a^\omega \cdot b \leq a^\omega$	—	—	—	13
Lm. 3.4.5(3)	$(a \cdot b)^\omega \leq (a + b)^\omega$	—	yes	—	1
Lm. 3.4.5(4)	$(a^+)^\omega = a^\omega$	—	yes	yes	4
Lm. 3.4.5(5)	$(a + b)^\omega = (a^* \cdot b)^\omega + (a^* \cdot b)^* \cdot a^\omega$	—	yes	yes	31
Lm. 3.4.5(5)	$(a + b)^\omega = (a^* \cdot b)^\omega + (a^* \cdot b)^* \cdot a^\omega$	yes	—	yes	0
Lm. 3.4.5(6)	$(a + b)^\omega = a^\omega + (a^* \cdot b) \cdot (a + b)^\omega$	—	—	yes	0
Lm. 3.4.5(6)	$(a + b)^\omega = a^\omega + (a^* \cdot b) \cdot (a + b)^\omega$	—	yes	yes	0
Pg. 51	$a \cdot 0 = a \Leftrightarrow \forall b : a \cdot b = a$	—	—	—	0
Lm. 3.4.7	$\mathsf{N} \cdot a \leq \mathsf{N}$ and $a \cdot \mathsf{N} \leq \mathsf{N}$	—	—	—	0
Lm. 3.4.9	$\mathsf{F} \cdot \mathsf{F} = \mathsf{F}$	—	—	—	0
Thm. 3.4.10	N and F exist in Boolean lazy semirings	—	—	—	2
Lm. 3.4.13	fin /inf commute with $\sqcap$	—	yes	yes	3
Lm. 3.4.14(1)	$a \cdot b = \mathsf{inf}\, a + \mathsf{fin}\, a \cdot b$	—	—	—	0
Lm. 3.4.14(2)	$\mathsf{inf}\,(a \cdot b) = \mathsf{inf}\, a + \mathsf{fin}\, a \cdot \mathsf{inf}\, b$	—	—	—	0
Lm. 3.4.14(3)	$\mathsf{fin}\,(a \cdot b) = \mathsf{fin}\,(\mathsf{fin}\, a \cdot b) \geq \mathsf{fin}\, a \cdot \mathsf{fin}\, b$	—	—	—	3
Lm. 3.4.14(4)	$\mathsf{fin}\,(\mathsf{fin}\, a \cdot b) = \mathsf{fin}\, a \cdot \mathsf{fin}\, b$	—	—	—	0
Lm. 3.4.15(1)	$a \leq \mathsf{F} \Leftrightarrow a = \mathsf{fin}\, a \Leftrightarrow \mathsf{inf}\, a = 0$	—	—	—	0
Lm. 3.4.15(1)	$a \leq \mathsf{N} \Leftrightarrow a = \mathsf{inf}\, a \Leftrightarrow \mathsf{fin}\, a = 0$	—	—	—	0

Table A.1: Summarised results using Prover9 (continued)

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[13]	**t[s]**
Lm. 3.4.15(2)	$a,b \leq \mathsf{F} \wedge c,d \leq \mathsf{N} \Rightarrow$ $(a+c \leq b+d \Leftrightarrow a \leq b \wedge c \leq d)$	—	—	—	5
Lm. 3.4.15(3)	$a^\omega = (\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + (\mathsf{fin}\, a)^\omega$	—	—	yes	12
Lm. 3.4.15(3)	$a^\omega = (\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + (\mathsf{fin}\, a)^\omega$	—	—	yes	1360
Lm. 3.4.15(4)	$\mathsf{inf}\, a^\omega = (\mathsf{fin}\, a)^* \cdot \mathsf{inf}\, a + \mathsf{inf}\, ((\mathsf{fin}\, a)^\omega)$	—	yes	—	0
Lm. 3.4.15(5)	$\mathsf{fin}\, a^\omega = (\mathsf{fin}\, a)^\omega \sqcap \mathsf{F} \leq (\mathsf{fin}\, a)^\omega$	—	yes	—	0
Lm. 3.5.2	test shunting	—	—	yes	8
Lm. 3.5.2	test shunting	—	—	yes	17
Lm. 3.5.3	distribution of test multiplication over meet (1st equation)	—	—	yes	0
Lm. 3.5.3	distribution of test multiplication over meet (1st equation)	—	yes	yes	3439
Lm. 3.5.3	distribution of test multiplication over meet (2nd equation)	—	yes	—	421
Cor. 3.5.4	$\overline{p \cdot a} = \neg p \cdot a + \overline{a}$	—	yes	yes	0
Cor. 3.5.4	$\overline{p \cdot a} = \neg p \cdot a + \overline{a}$	—	yes	yes	35
Lm. 3.5.5(1)	$p \leq q \Leftrightarrow p \cdot \top \leq q \cdot \top$	—	—	—	197
Lm. 3.5.5(2)	$p^* = 1$	—	—	—	0
Lm. 3.5.5(2)	$p^\omega = p \cdot \top$	—	—	—	8
Lm. 3.5.5(3)	$p \cdot (p \cdot a)^\omega = (p \cdot a)^\omega$	—	—	yes	5
Lm. 3.5.5(3)	$(p \cdot a)^\omega = (p \cdot a \cdot p)^\omega$	—	—	yes	5
Lm. 3.6.2	(d1) $\wedge$ (d2) $\Leftrightarrow$ (llp)	—	—	yes	0
Lm. 3.6.2	(d1) $\wedge$ (d2) $\Leftrightarrow$ (llp)	—	—	yes	0

Table A.1: Summarised results using Prover9 (continued)

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[13]	**t[s]**
Lm. 3.6.2	$(\mathrm{d1}) \wedge (\mathrm{d2}) \Leftrightarrow (\mathrm{gla})$	—	—	yes	6
Lm. 3.6.2	$(\mathrm{d1}) \wedge (\mathrm{d2}) \Leftrightarrow (\mathrm{gla})$	—	—	yes	278
Lm. 3.6.3(1)	domain is isotone	—	—	—	50
Lm. 3.6.3(2)	$\lceil 0 = 0$ and $\lceil(a+b) = \lceil a + \lceil b$	—	yes	yes	86
Lm. 3.6.3(2)	$\lceil a + \lceil \overline{a} = 1$	—	yes	—	0
Lm. 3.6.3(3)	domain is fully strict	—	—	—	0
Lm. 3.6.3(4)	$\lceil p = p$	—	—	—	0
Lm. 3.6.3(5)	import/export rule	—	yes	—	18
Lm. 3.6.3(6)	$\lceil(a \cdot b) \leq \lceil a$	—	—	—	1
Lm. 3.6.4	$\lceil a \leq p \Leftrightarrow a \leq p \cdot \top$	—	—	—	18
Lm. 3.6.6	$(\mathrm{cd1}) \wedge (\mathrm{cd2}) \Leftrightarrow (\mathrm{lrp})$	—	—	yes	0
Lm. 3.6.6	$(\mathrm{cd1}) \wedge (\mathrm{cd2}) \Leftrightarrow (\mathrm{lrp})$	—	—	yes	0
Lm. 3.6.6	$(\mathrm{cd1}) \wedge (\mathrm{cd2}) \Leftrightarrow (\mathrm{wgra})$	—	—	yes	0
Lm. 3.6.6	$(\mathrm{cd1}) \wedge (\mathrm{cd2}) \Leftrightarrow (\mathrm{wgra})$	—	—	yes	1
Lm. 3.6.7(1)	codomain is isotone	—	—	—	0
Lm. 3.6.7(2)	$\overline{0}^{\rceil} = 0$ and $(a+b)^{\rceil} = a^{\rceil} + b^{\rceil}$	—	yes	yes	1
Lm. 3.6.7(2)	$a^{\rceil} + \overline{a}^{\rceil} = 1$	—	yes	—	0
Lm. 3.6.7(3)	$a^{\rceil} \leq 0 \Leftrightarrow a \leq \mathsf{N}$	—	—	—	0
Lm. 3.6.7(4)	$p^{\rceil} = p$	—	—	—	0
Lm. 3.6.7(5)	import/export rule	—	—	—	4
Lm. 3.6.7(6)	$(a \cdot b)^{\rceil} \leq b^{\rceil}$	—	—	—	1

Table A.1: Summarised results using Prover9 (continued)

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[13]	**t[s]**
Lm. 3.6.10(1)	$a \leq p \cdot a \Leftrightarrow a \leq p \cdot \top$	—	—	—	0
Lm. 3.6.10(2)	$a \leq a \cdot p \Leftrightarrow a \leq \top \cdot p$	—	—	—	3
Lm. 3.6.10(3)	$a \leq \mathsf{F} \Leftrightarrow (a \leq a \cdot p \Leftrightarrow a \cdot \neg p \leq 0)$	—	yes	—	25
Lm. 3.6.10(3)	$a \leq \mathsf{F} \Leftrightarrow (a \leq \top \cdot p \Leftrightarrow a \cdot \neg p \leq 0)$	—	yes	—	122
Lm. 4.1.2	basic properties of time domain	—	—	—	0
Lm. 4.1.4	∞ is greatest element and unique	—	—	—	0
Lm. 4.4.1	$a^\omega = a$ if a is purely infinite	—	—	—	0
Lm. 4.4.9	every element is convergent	—	—	—	0
Lm. 4.5.2(1)	c submodular $\Rightarrow (c \sqcap \mathsf{F}) \cdot \overline{c} + \overline{c} \cdot \top \leq \overline{c}$	—	—	yes	33
Lm. 4.5.2(1)	$(c \sqcap \mathsf{F}) \cdot \overline{c} + \overline{c} \cdot \top \leq \overline{c} \Rightarrow \mathsf{F} \cdot \overline{c} \cdot \top \leq \overline{c}$	—	—	yes	200
Lm. 4.5.2(1)	$\mathsf{F} \cdot \overline{c} \cdot \top \leq \overline{c} \Rightarrow (c \sqcap \mathsf{F}) \cdot \overline{c} + \overline{c} \cdot \top \leq \overline{c}$	—	—	yes	9
Lm. 4.5.2(1)	$(c \sqcap \mathsf{F}) \cdot \overline{c} + \overline{c} \cdot \top \leq \overline{c} \Rightarrow c$ submodular	yes	yes	yes	$93{+}2^{14}$
Lm. 4.5.2(2)	c modular iff c submodular and transitive	—	—	yes	0
Lm. 4.5.2(2)	c modular iff c submodular and transitive	—	—	yes	318
Lm. 4.5.2(3)	submodular elements are closed under $+$	—	yes	—	4
Lm. 4.5.2(3)	submodular elements are closed under $\sqcap$	yes	yes	—	7
Lm. 4.5.2(4)	$c \sqcap a^+ = (c \sqcap a)^+$	—	—	yes	6
Lm. 4.5.2(4)	$c \sqcap a^+ = (c \sqcap a)^+$	yes	yes	yes	$41{+}82^{14}$
Lm. 4.5.2(5)	d submodular iff $(c \sqcap \mathsf{F}) \cdot (c \sqcap \overline{d}) \cdot c \leq c \sqcap \overline{d}$	—	—	yes	0
Lm. 4.5.2(5)	d submodular iff $(c \sqcap \mathsf{F}) \cdot (c \sqcap \overline{d}) \cdot c \leq c \sqcap \overline{d}$	—	—	yes	0

[14] We had to show the goal in two steps.

Table A.1: Summarised results using Prover9 (continued)

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[13]	**t[s]**
Lm. 4.5.7	tests modular iff S is progressive	—	—	yes	7
Lm. 4.5.7	tests modular iff S is progressive	—	yes	yes	271
Pg. 88	$\diamondsuit 0 = 0 = \Box 0$, $\diamondsuit 1 = \top = \Box 1$	—	—	—	4
Lm. 4.5.7	tests are modular iff S progressive	—	—	yes	7
Lm. 4.5.7	tests are modular iff S progressive	—	yes	yes	260
Lm. 4.5.9	$\Box p$ is submodular	—	yes	—	350
Lm. 4.5.10(1)	$p \leq \Box q \Leftrightarrow p \leq q$	—	yes	yes	16
Lm. 4.5.10(1)	$p \leq \Box q \Leftrightarrow p \leq q$	—	yes	yes	1105
Lm. 4.5.10(2)	$p \leq \Box p$	—	yes	—	13
Lm. 4.5.11	$\Box p = p \cdot (\Box p)$	—	yes	—	142
Lm. 4.5.11	$\Box p = p \cdot (\Box p)$	—	—	—	1609
Lm. 4.5.13(1)	all boxes are modular	—	yes	—	0
Lm. 4.5.13(2)	$(\Box p) \cdot (\Box p) = \Box p$	—	yes	—	9
Lm. 4.5.13(3)	$\Box p \sqcap a^+ = (\Box p \sqcap a)^+$	—	yes	—	0
Lm. 4.5.13(4)	$\diamondsuit p \sqcap a \cdot b = (\diamondsuit p \sqcap a) \cdot b + a^\frown(\diamondsuit p \sqcap b)$	yes	yes	yes	0
Lm. 4.5.13(4)	$\diamondsuit p \sqcap a \cdot b = (\diamondsuit p \sqcap a) \cdot b + a^\frown(\diamondsuit p \sqcap b)$	yes	yes	yes	60
Lm. 4.5.17	$a \lfloor b \leq x \Leftrightarrow \overline{x} \cdot b \leq \overline{a}$	—	—	—	133
Lm. 4.5.19	local linearity implies safety-closedness	yes	yes	—	224
Cor. 4.5.23	$a \leq \Box(\mathsf{ran}\ a)$, $\mathsf{ran}\ (\Box p) \leq p$ and $p \leq \Box p \Rightarrow \mathsf{ran}\ p \leq p$	—	—	—	1
Lm. 4.5.24	$\mathsf{ran}\ p = p$	—	yes	—	0
Lm. 4.5.25	$\diamondsuit p \cdot \diamondsuit p = \diamondsuit p$	—	—	yes	0

Table A.1: Summarised results using Prover9 (continued)

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[13]	**t[s]**
Lm. 4.5.25	$\diamondsuit p \cdot \diamondsuit p = \diamondsuit p$	—	—	yes	126
Lm. 4.6.3	$\lceil(\Box p) = p$	—	yes	—	3
Lm. 4.6.4	$\lceil(b \sqcap w) = w \lfloor b \sqcap 1$	—	—	yes	0
Lm. 4.6.4	$\lceil(b \sqcap w) = w \lfloor b \sqcap 1$	—	yes	yes	0
Lm. 4.6.6(1)	$\mathsf{E}a.(v + w) = \mathsf{E}a.v + \mathsf{E}a.w$	—	yes	yes	142
Lm. 4.6.6(1)	$\mathsf{E}a.(v + w) = \mathsf{E}a.v + \mathsf{E}a.w$	—	—	yes	2
Lm. 4.6.6(1)	$\mathsf{A}a.(v \sqcap w) = \mathsf{A}a.v \sqcap \mathsf{A}a.w$	—	—	yes	55
Lm. 4.6.6(1)	$\mathsf{A}a.(v \sqcap w) = \mathsf{A}a.v \sqcap \mathsf{A}a.w$	—	—	yes	4
Lm. 4.6.6(2)	$\mathsf{E}(a \cdot b)\,.\,c = \mathsf{E}a\,.\,(\mathsf{E}b\,.\,c)$	—	—	yes	0
Lm. 4.6.6(2)	$\mathsf{E}(a \cdot b)\,.\,c = \mathsf{E}a\,.\,(\mathsf{E}b\,.\,c)$	—	—	yes	0
Lm. 4.6.6(2)	$\mathsf{A}(a \cdot b)\,.\,c = \mathsf{A}a\,.\,(\mathsf{A}b\,.\,c)$	—	—	yes	0
Lm. 4.6.6(2)	$\mathsf{A}(a \cdot b)\,.\,c = \mathsf{A}a\,.\,(\mathsf{A}b\,.\,c)$	—	—	yes	0
Lm. 5.2.10(1)	$\forall b \in S : \mathsf{n} \cdot \bar{b} \leq \overline{\mathsf{n} \cdot b} \Leftrightarrow$ $\forall b, c \in S : \mathsf{n} \cdot (b \sqcap c) = \mathsf{n} \cdot b \sqcap \mathsf{n} \cdot c$	—	—	yes	23
Lm. 5.2.10(1)	$\forall b \in S : \mathsf{n} \cdot \bar{b} \leq \overline{\mathsf{n} \cdot b} \Leftrightarrow$ $\forall b, c \in S : \mathsf{n} \cdot (b \sqcap c) = \mathsf{n} \cdot b \sqcap \mathsf{n} \cdot c$	—	yes	yes	4
Lm. 5.2.10(1)	$\forall b \in S : \mathsf{n} \cdot \bar{b} \leq \overline{\mathsf{n} \cdot b} \Leftrightarrow$ $\forall b, c \in S : \mathsf{n} \cdot (b \sqcap c) = \mathsf{n} \cdot b \sqcap \mathsf{n} \cdot c$	—	yes	yes	1
Lm. 5.2.10(2)	$\forall b \in S : \overline{\mathsf{n} \cdot b} \leq \mathsf{n} \cdot \bar{b} \Leftrightarrow \mathsf{n} \cdot \top = \top$	—	—	—	7
Lm. 5.2.10(2)	$\mathsf{n} \cdot \top = \top \Leftrightarrow \mathsf{n}^\omega = \top$	—	—	—	1
Lm. 5.4.2	$\overrightarrow{\mathsf{n}}_l b = \top \cdot \lceil b$	—	—	—	1
Lm. 5.4.2	$\overrightarrow{\mathsf{n}}_r b = \bar{b}\rceil \cdot \top$	—	—	—	2
Lm. 5.4.2	$\overleftarrow{\mathsf{n}}_l b = \lceil b \cdot \top$	—	—	—	0

Table A.1: Summarised results using Prover9 (continued)

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[13]	**t[s]**
Lm. 5.4.2	$\langle\!\rangle_r b = \top \cdot \overline{b}$	—	—	—	0
Cor. 5.4.3	$\langle\!\rangle_l \langle\!\rangle_r b = \langle\!\rangle_r b$	—	—	—	1
Cor. 5.4.3	$\langle\!\rangle_r \langle\!\rangle_l b = \langle\!\rangle_l b$	—	—	—	1
Cor. 5.4.3	$\langle\!\rangle_l \langle\!\rangle_r b = \langle\!\rangle_r b$	—	—	—	1
Cor. 5.4.3	$\langle\!\rangle_r \langle\!\rangle_l b = \langle\!\rangle_l b$	—	—	—	1
Cor. 5.4.3	$\langle\!\rangle_l \langle\!\rangle_l b = \langle\!\rangle_l b$	—	—	—	1
Cor. 5.4.3	$\langle\!\rangle_r \langle\!\rangle_r b = \langle\!\rangle_r b$	—	—	—	0
Cor. 5.4.3	$\langle\!\rangle_l \langle\!\rangle_l b = \langle\!\rangle_l b$	—	—	—	1
Cor. 5.4.3	$\langle\!\rangle_r \langle\!\rangle_r b = \langle\!\rangle_r b$	—	—	—	1
Lm. 5.4.5	$[\!]_l b \leq \langle\!\rangle_l b$	—	yes	—	44
Lm. 5.4.5	$[\!]_r b \leq \langle\!\rangle_r b$	—	yes	—	50
Lm. 5.4.5	$[\!]_l b \leq \langle\!\rangle_l b$	—	yes	—	53
Lm. 5.4.5	$[\!]_r b \leq \langle\!\rangle_r b$	—	yes	—	55
Cor. 5.4.6	$a \leq [\!]_l b \Leftrightarrow \overline{b} \leq [\!]_r \overline{a}$	—	—	—	10
Lm. 5.4.7	$[\!]_l b = \top \cdot \neg\overline{b}$	yes	yes	—	4
Lm. 5.4.7	$[\!]_r b = \neg\overline{b} \cdot \top$	yes	yes	—	5
Lm. 5.4.7	$[\!]_l b = \neg\overline{b} \cdot \top$	yes	yes	—	4
Lm. 5.4.7	$[\!]_r b = \top \cdot \neg\overline{b}$	yes	yes	—	4
Lm. 5.4.8(1)	$\overline{\langle\!\rangle_r \overline{b}} = [\!]_r b$	—	yes	—	0
Lm. 5.4.8(1)	$\overline{[\!]_r \overline{b}} = \langle\!\rangle_r b$	—	yes	—	0
Lm. 5.4.8(2)	$\overline{\langle\!\rangle_l \overline{b}} = [\!]_l b$	—	yes	—	0

Table A.1: Summarised results using Prover9 (continued)

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[13]	**t[s]**
Lm. 5.4.8(2)	$\overline{\Box_l \bar{b}} = \Diamond_l b$	—	yes	—	0
Lm. 5.4.8(3)	$\overline{\Diamond_l \bar{b}} \leq \Box_l b$	—	yes	—	536
Lm. 5.4.8(3)	$\overline{\Box_l \bar{b}} \leq \Diamond_l b$	—	yes	—	1035
Lm. 5.4.8(4)	$\overline{\Diamond_r \bar{b}} \leq \Box_r b$	—	yes	—	521
Lm. 5.4.8(4)	$\overline{\Box_r \bar{b}} \leq \Diamond_r b$	—	yes	—	969
Thm. 5.4.9	$\Diamond_r a \leq b \Leftrightarrow a \leq \Box_l b$	—	yes	yes	118
Thm. 5.4.9	$\Diamond_r a \leq b \Leftrightarrow a \leq \Box_l b$	—	yes	yes	0
Thm. 5.4.9	$\Diamond_l a \leq b \Leftrightarrow a \leq \Box_l b$	—	yes	yes	0
Thm. 5.4.9	$\Diamond_l a \leq b \Leftrightarrow a \leq \Box_l b$	—	yes	yes	0
Cor. 5.4.10(1)	$\Diamond_r$ is isotone	—	—	—	0
Cor. 5.4.10(1)	$\Diamond_l$ is isotone	—	—	—	7
Cor. 5.4.10(1)	$\Box_l$ is isotone	—	—	—	113
Cor. 5.4.10(1)	$\Box_l$ is isotone	—	—	—	178
Cor. 5.4.10(2)	$\Diamond_r$ is disjunctive	—	—	yes	0
Cor. 5.4.10(2)	$\Diamond_l$ is disjunctive	—	yes	—	1
Cor. 5.4.10(2)	$\Diamond_l$ is disjunctive	—	yes	—	0
Cor. 5.4.10(2)	$\Box_l$ is conjunctive	—	yes	yes	2
Cor. 5.4.10(2)	$\Box_l$ is conjunctive	—	yes	yes	0
Cor. 5.4.10(2)	$\Box_l$ is conjunctive	—	yes	yes	0
Cor. 5.4.10(3)	$\Diamond_r \Box_l a \leq a \leq \Box_l \Diamond_r a$	—	yes	—	0
Cor. 5.4.10(3)	$\Diamond_l \Box_l a \leq a \leq \Box_l \Diamond_l a$	—	yes	—	0

Table A.1: Summarised results using Prover9 (continued)

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[13]	**t[s]**
Thm 5.4.11	$\diamond_l a \leq b \Rightarrow a \leq \boxdot_r b$	—	yes	—	166
Thm 5.4.11	$\diamond_r a \leq b \Rightarrow a \leq \boxdot_r b$	—	yes	—	135
Lm. 5.4.12(1)	$\diamond_l$ is isotone	—	yes	—	0
Lm.5.4.12(1)	$\diamond_r$ is isotone	—	—	—	0
Lm. 5.4.12(1)	$\boxdot_r$ is isotone	—	—	—	182
Lm. 5.4.12(1)	$\boxdot_r$ is isotone	—	—	—	333
Lm. 5.4.12(2)	$\diamond_l$ is disjunctive	—	yes	—	0
Lm. 5.4.12(2)	$\diamond_r$ is disjunctive	—	yes	—	0
Lm. 5.4.12(2)	$\boxdot_r$ is conjunctive	—	yes	—	0
Lm. 5.4.12(2)	$\boxdot_r$ is conjunctive	—	yes	—	0
Lm. 5.4.13(1)	$\cdot 1 = \cdot \top = \boxdot \top = \top$	—	yes	—	3
Lm. 5.4.13(2)	$\diamond_r 0 = \diamond_l 0 = \boxdot_r 0 = \boxdot_l 0 = 0$, $\diamond_l 0 = \diamond_r 0 = \boxdot_l 0 = \boxdot_r 0 = \mathsf{N}$	—	yes	—	4
Lm. 5.4.13(3)	$\ulcorner a \leq \cdot_l a$, $\bar{a}\urcorner \leq \cdot_r a$, $a \leq \diamond_l a$, $a \leq \diamond_r$	—	—	—	0
Lm. 5.4.14(1)	$\diamond_l \mathsf{F} = \diamond_r \mathsf{F} = \diamond_l \mathsf{F} = \diamond_r \mathsf{F} = \top$	—	yes	—	1
Lm. 5.4.14(2)	$b \leq \mathsf{N} \Leftrightarrow \diamond_r b \leq 0$	—	—	—	1
Lm. 5.4.14(2)	$\diamond_r b \leq 0 \Leftrightarrow \diamond_r b \leq \mathsf{N}$	—	—	—	0
Lm. 5.4.14(2)	$\diamond_r b \leq 0 \Leftrightarrow \diamond_r b \leq \mathsf{N}$	—	—	—	0
Lm. 5.4.14(3)	$\boxdot_l \mathsf{N} = \boxdot_r \mathsf{N} = \mathsf{N}$, $\boxdot_r \mathsf{N} = \boxdot_l \mathsf{N} = 0$	—	—	—	62
Lm. 5.4.14(4)	$\bar{b} \leq \mathsf{N} \Leftrightarrow \mathsf{F} \leq b$	—	—	—	0
Lm. 5.4.14(4)	$\mathsf{F} \leq b \Leftrightarrow \boxdot_r b = \top$	—	—	—	9
Lm. 5.4.14(4)	$\boxdot_r b = \top \Leftrightarrow \boxdot_r b = \top$	—	—	—	39

Table A.1: Summarised results using Prover9 (continued)

Nr.	**Theorem**	**removed axioms**	**additional hypotheses**	**split**[13]	**t[s]**
Pg. 123	$\langle b \rangle_l \boxplus_l = \boxplus_l b$	—	—	yes	0
Lm. 5.5.2	$\langle ? \rangle_l b = \mathsf{E} b \,.\, \top$	—	yes	—	0
Case Study 6.2	route planning I $\tau \leq AT_u \cdot (AT_u \cdot A_5 \cdot AT_d + AT_d \cdot A_5 \cdot AT_u + AT_d \cdot A_{10} \cdot AT_b + AT_b \cdot A_{10} \cdot AT_d + AT_b \cdot A_{15} \cdot AT_u + AT_u \cdot A_{15} \cdot AT_b)^\omega$	yes	yes	yes	7
Case Study 6.2	route planning IIa $(AT_u \cdot A_5 \cdot AT_d \cdot A_{10} \cdot AT_b \cdot A_{10})^\omega \leq (A_{30} \cdot AT_u)^\omega$	yes	yes	—	0
Case Study 6.2	route planning IIb $(AT_u \cdot A_5 \cdot AT_d \cdot A_{10} \cdot AT_b \cdot A_{10})^\omega \leq (A_{30} \cdot AT_d)^\omega$	yes	yes	—	0
Case Study 6.2	route planning IIc $(AT_u \cdot A_5 \cdot AT_d \cdot A_{10} \cdot AT_b \cdot A_{10})^\omega \leq (A_{30} \cdot AT_b)^\omega$	yes	yes	—	0
Case Study 6.3	assembly line scheduler $((I_D \cdot L_1 \cdot I_D + I_D \cdot L_2 \cdot I_D) \sqcap A_{=4})^\omega \leq (\mathsf{F} \cdot L_1)^\omega$	yes	yes	—	27

Appendix B

Deferred Properties and Proofs

Quand on veut un mouton,
c'est la preuve qu'on existe.
A. de Saint Exupéry (Le Petit Prince)

This appendix gives additional details concerning algebraic properties. In particular we list a couple of properties for the box operator $\Box p$. Moreover, we lists many of the deferred proofs of the thesis.

B.1 Additional Properties

Lemma B.1.1 *Assume that S is a Boolean and safety-closed weak semiring. Then, for all $p, q \in \mathsf{test}(S)$, we have*

1. $\Box p \sqcap q \leq p \cdot q$.
2. $p \leq \Box p \;\Rightarrow\; p \cdot q \leq \Box p \sqcap q$.
3. $\Box p \sqcap \diamondsuit q \;=\; \mathbb{F} p \cdot q \cdot \Box p$.
4. $\mathbb{F} p \sqcap \diamondsuit q \;=\; \mathbb{F} p \cdot \neg q \cdot \mathbb{F} p$.
5. $\Box p \sqcap \overline{\mathbb{F} q} \;=\; \mathbb{N} p + \mathbb{F} p \cdot q \cdot \Box p$.
6. $\Box p \sqcap \Box q \leq \Box p \cdot \Box q$.
7. $\mathbb{F} p \sqcap \Box q \leq \mathbb{F} p \cdot \Box q$ *and* $\mathbb{F} p \sqcap \Box q \leq \Box q \cdot \mathbb{F} p$.

Proof.

1. $\Box p \sqcap q \;=\; (\Box p) \cdot p \sqcap q \;=\; (\Box p \sqcap q) \cdot p \leq q \cdot p$.
2. $p \cdot q \;=\; p \sqcap q \leq \Box p \sqcap q$.
3. $\Box p \sqcap \diamondsuit q \;=\; \Box p \sqcap (\mathsf{F} \cdot q \cdot \top) \;=\; (\Box p \sqcap \mathsf{F}) \cdot (\Box p \sqcap q) \cdot (\Box p \sqcap \top) \;=\; \mathbb{F} p \cdot p \cdot q \cdot \Box p \;=\; \mathbb{F} p \cdot q \cdot \Box p$.
4. Analogously.
5. $\Box p \sqcap \overline{\mathbb{F} q} \;=\; \Box p \sqcap (\mathsf{N} + \diamondsuit \neg q) \;=\; \mathbb{N} p + \mathbb{F} p \cdot \neg q \cdot \Box p$ by distributivity and previous result.
6. $\Box p \sqcap \Box q \;=\; \Box p \sqcap \Box q \cdot \Box q \;=\; (\Box p \sqcap \Box q) \cdot (\Box p \sqcap \Box q) \leq \Box p \cdot \Box q$.
7. Analogously. □

Lemma B.1.2 *Assume that S is a Boolean and weak locally linear quantale. Then, for all $a \in S$, $p, q \in \mathsf{test}(S)$, we have*

1. $(\Box p \cdot \Box q) \lfloor a \leq \Box p \cdot \Box q + \Box p$.
2. $(\mathbb{F} p \cdot \Box q) \lfloor a \leq \mathbb{F} p \cdot \Box q + \mathbb{F} p$.

Similarly to to the detachment $\lfloor$, one can define $a \rfloor b$ as $\overline{a \backslash \overline{b}}$ for purely finite *a, where $a \cdot x \leq b \;\Leftrightarrow\; x \leq a \backslash b$. The* left detachment *exists, since $a \cdot$ is fully disjunctive if a is purely finite. Moreover $\rfloor$ is the lower adjoint of a Galois connection.*

3. *If a is purely finite, then $a \rfloor (\Box p \cdot \Box p) \leq \Box p \cdot \Box q + \Box q$.*
4. *If a is purely finite, then $a \rfloor (\mathbb{F} p \cdot \Box p) \leq \mathbb{F} p \cdot \Box q + \Box q$.*
5. $a \cdot b \sqcap \mathbb{F} p \cdot \Box q \leq (a \sqcap \mathbb{F} p \cdot \Box q) \cdot b + (a \sqcap \mathbb{F} p) \cdot b$.
6. *If a is purely finite then $a \cdot b \sqcap \mathbb{F} p \cdot \Box q \leq a \cdot (b \sqcap \mathbb{F} p \cdot \Box q) + a \cdot (b \sqcap \Box q)$.*

Proof.

1. By local linearity and box detachment twice,

$$(\Box p \cdot \Box q) \lfloor a \;=\; \Box p \cdot (\Box q \lfloor a) + \Box p \lfloor (a \lfloor \Box q) \leq \Box p \cdot \Box q + \Box p \;.$$

2. Analogously.
3. Analogously, using $a \leq \mathsf{F}$.
4. Analogously.

5. In a Boolean quantale, we have the Dedekind rule

$$u \cdot v \sqcap w \leq (u \sqcap w \lfloor v) \cdot v \ .$$

Using this, part (2) and distributivity we obtain

$$a \cdot b \sqcap \mathbb{F} p \cdot \Box q \leq (a \sqcap (\mathbb{F} p \cdot \Box q) \lfloor b) \cdot b \leq (a \sqcap \mathbb{F} p \cdot \Box q) \cdot b + (a \sqcap \mathbb{F} p) \cdot b \ .$$

6. Analogously. However, in contrast to full quantales, in Boolean left quantales the dual Dedekind rule

$$u \cdot v \sqcap w \leq u \cdot (v \sqcap u \rfloor w)$$

holds only for purely finite u. □

B.2 Deferred Proofs

Theorem 4.1.10

1. The processes under union as addition and composition as multiplication form a Boolean weak quantale $\mathsf{PRO} =_{df} (\mathcal{P}(\mathsf{TRA}), \cup, \cdot, \emptyset, I)$, where I is the set of all zero-duration trajectories, i.e., $I = \{\underline{x} \,|\, x \in V\}$.
2. Setting $\mathsf{test}(\mathsf{PRO})$ as $\mathcal{P}(I)$, PRO becomes a test weak quantale. In particular, multiplication $(\cdot)$ is universally disjunctive in its right argument.
3. Additionally, chop inherits the disjunctivity properties from $\cdot$ and is associative.

Proof. We assume A, B, C and A_i to be arbitrary processes.

1. We first verify the Axioms (3.1)–(3.9) for processes. This implies that PRO forms a weak semiring.

   - $(\mathcal{P}(\mathsf{TRA}), \cup, \emptyset)$ forms a commutative monoid. (Axioms (3.1)-(3.3)) Since PRO is based on sets, this is straightforward.
   - $(\mathcal{P}(\mathsf{TRA}), \cdot, I)$ forms a monoid.
     Axiom (3.5) (associativity):
     The purely finite and infinite parts of a process are defined as follows (cf. Page 64):

$$\mathsf{fin}\, A =_{df} \{(d, g) \,|\, (d, g) \in A,\ d \neq \infty\} \quad \text{and}$$
$$\mathsf{inf}\, A =_{df} \{(d, g) \,|\, (d, g) \in A,\ d = \infty\} \ .$$

     This definition implies immediately $\mathsf{fin}\,(\mathsf{fin}\, A) = \mathsf{fin}\, A$, $\mathsf{inf}\,(\mathsf{inf}\, A) = \mathsf{inf}\, A$, $\mathsf{fin}\,(A \cup B) = \mathsf{fin}\,(A) \cup \mathsf{fin}\,(B)$, $\mathsf{inf}\,(A \cup B) = \mathsf{inf}\,(A) \cup \mathsf{inf}\,(B)$, $\mathsf{fin}\,(\mathsf{inf}\, A) = \emptyset$ and $\mathsf{inf}\,(\mathsf{fin}\, A) = \emptyset$.

$$
\begin{aligned}
& (A \cdot B) \cdot C \\
= \quad & \{\![\text{ Definition 4.1.9 }]\!\} \\
& (\mathsf{inf}\, A \cup \{a \cdot b \mid a \in \mathsf{fin}\, A,\, b \in B,\, a \cdot b \text{ def.}\}) \cdot C \\
= \quad & \{\![\text{ Definition 4.1.9 }]\!\} \\
& \mathsf{inf}\, (\mathsf{inf}\, A \cup \{a \cdot b \mid a \in \mathsf{fin}\, A,\, b \in B,\, a \cdot b \text{ def.}\}) \cup \\
& \{d \cdot c \mid d \in \mathsf{fin}\, (\mathsf{inf}\, A \cup \{a \cdot b \mid a \in \mathsf{fin}\, A,\, b \in B,\, a \cdot b \text{ def.}\}), \\
& \qquad c \in C,\, d \cdot c \text{ def.}\} \\
= \quad & \{\![\text{ above remark }]\!\} \\
& \mathsf{inf}\, (\mathsf{inf}\, A) \cup \mathsf{inf}\, (\{a \cdot b \mid a \in \mathsf{fin}\, A,\, b \in B,\, a \cdot b \text{ def.}\}) \cup \\
& \{d \cdot c \mid d \in \mathsf{fin}\, (\mathsf{inf}\, A) \cup \mathsf{fin}\, (\{a \cdot b \mid a \in \mathsf{fin}\, A,\, b \in B,\, a \cdot b \text{ def.}\}), \\
& \qquad c \in C,\, d \cdot c \text{ def.}\} \\
= \quad & \{\![\text{ above remark }]\!\} \\
& \mathsf{inf}\, A \cup \{a \cdot b \mid a \in \mathsf{fin}\, A,\, b \in \mathsf{inf}\, B,\, a \cdot b \text{ def.}\} \cup \\
& \{d \cdot c \mid d \in \{a \cdot b \mid a \in \mathsf{fin}\, A,\, b \in \mathsf{fin}\, B,\, a \cdot b \text{ def.}\},\, c \in C,\, d \cdot c \text{ def.}\} \\
= \quad & \{\![\text{ set theory }]\!\} \\
& \mathsf{inf}\, A \cup \{a \cdot b \mid a \in \mathsf{fin}\, A,\, b \in \mathsf{inf}\, B,\, a \cdot b \text{ def.}\} \cup \\
& \{(a \cdot b) \cdot c \mid a \in \mathsf{fin}\, A,\, b \in \mathsf{fin}\, B,\, c \in C,\, a \cdot b \text{ and } (a \cdot b) \cdot c \text{ def.}\} \\
= \quad & \{\![\text{ associativity of composition of trajectories (Page 63) }]\!\} \\
& \mathsf{inf}\, A \cup \{a \cdot b \mid a \in \mathsf{fin}\, A,\, b \in \mathsf{inf}\, B,\, a \cdot b \text{ def.}\} \cup \\
& \{a \cdot (b \cdot c) \mid a \in \mathsf{fin}\, A,\, b \in \mathsf{fin}\, B,\, c \in C,\, b \cdot c \text{ and } a \cdot (b \cdot c) \text{ def.}\} \\
= \quad & \{\![\text{ set theory }]\!\} \\
& \mathsf{inf}\, A \cup \{a \cdot d \mid a \in \mathsf{fin}\, A,\, d \in \mathsf{inf}\, B,\, a \cdot d \text{ def.}\} \cup \\
& \{a \cdot d \mid a \in \mathsf{fin}\, A,\, d \in \{b \cdot c \mid b \in \mathsf{fin}\, B,\, c \in C,\, b \cdot c \text{ def.}\},\, a \cdot d \text{ def.}\} \\
= \quad & \{\![\text{ set theory }]\!\} \\
& \mathsf{inf}\, A \cup \{a \cdot d \mid a \in \mathsf{fin}\, A,\, d \in \mathsf{inf}\, B \cup \{b \cdot c \mid b \in \mathsf{fin}\, B,\, c \in C,\, b \cdot c \text{ def.}\}, \\
& \qquad a \cdot d \text{ def.}\} \\
= \quad & \{\![\text{ Definition 4.1.9 }]\!\} \\
& A \cdot (\mathsf{inf}\, B \cup \{b \cdot c \mid b \in \mathsf{fin}\, B,\, c \in C,\, b \cdot c \text{ def.}\}) \\
= \quad & \{\![\text{ Definition 4.1.9 }]\!\} \\
& A \cdot (B \cdot C)
\end{aligned}
$$

Axioms (3.5) and (3.6) (neutral element):

$$
\begin{aligned}
& A \cdot I \\
= \quad & \{\![\text{ Definition 4.1.9 and definition of } I \text{ }]\!\} \\
& \mathsf{inf}\, A \cup \{a \cdot \underline{v} \mid a \in \mathsf{fin}\, A,\, v \in V,\, a \cdot \underline{v} \text{ def.}\} \\
= \quad & \{\![\; (\subseteq)\; a \cdot \underline{v} \leq a \text{ if defined (Page 63)}, \\
& \quad\; (\supseteq) \text{ for each } a = (g, d) \text{ choose } v \text{ as } g(d) \text{ }]\!\}
\end{aligned}
$$

$$
\begin{array}{ll}
 & \mathsf{inf}\, A \cup \{a \mid a \in \mathsf{fin}\, A\} \\
= & \quad \{\!\![\text{ set theory }]\!\!\} \\
 & \mathsf{inf}\, A \cup \mathsf{fin}\, A \\
= & \quad \{\!\![\text{ definition of } \mathsf{fin}/\mathsf{inf}]\!\!\} \\
 & A
\end{array}
$$

$I \cdot A = A$ can be shown in a similar way using the fact that I is purely finite and therefore $\mathsf{inf}\, I = \emptyset$.

- Axioms (3.7) and (3.8) (distributivity):
  We show the more general claim that $\cdot$ is positively disjunctive in both arguments. The set-based definition of purely (in)finite parts of processes implies that $\mathsf{inf}\,(\bigcup_{i\in I} A_i) = \bigcup_{i\in I} \mathsf{inf}\, A_i$ and $\mathsf{fin}\,(\bigcup_{i\in I} A_i) = \bigcup_{i\in I} \mathsf{fin}\, A_i$.

$$
\begin{array}{ll}
 & \bigcup_{i\in I} A_i \cdot C \\
= & \quad \{\!\![\text{ Definition 4.1.9 }]\!\!\} \\
 & \mathsf{inf}\,(\bigcup_{i\in I} A_i) \cup \{a \cdot c \mid a \in \mathsf{fin}\,(\bigcup_{i\in I} A_i),\, c \in C,\, a \cdot c \text{ def.}\} \\
= & \quad \{\!\![\text{ above remark }]\!\!\} \\
 & \bigcup_{i\in I} \mathsf{inf}\, A_i \cup \{a \cdot c \mid a \in \bigcup_{i\in I} \mathsf{fin}\, A_i,\, c \in C,\, a \cdot c \text{ def.}\} \\
= & \quad \{\!\![\text{ set theory }]\!\!\} \\
 & \bigcup_{i\in I} \mathsf{inf}\, A_i \cup \bigcup_{i\in I} \{a \cdot c \mid a \in \mathsf{fin}\, A_i,\, c \in C,\, a \cdot c \text{ def.}\} \\
= & \quad \{\!\![\text{ set theory }]\!\!\} \\
 & \bigcup_{i\in I} (\mathsf{inf}\, A_i \cup \{a \cdot c \mid a \in \mathsf{fin}\, A_i,\, c \in C,\, a \cdot c \text{ def.}\}) \\
= & \quad \{\!\![\text{ Definition 4.1.9 }]\!\!\} \\
 & \bigcup_{i\in I} (A_i \cdot C)
\end{array}
$$

$$
\begin{array}{ll}
 & C \cdot \bigcup_{i\in I} A_i \\
= & \quad \{\!\![\text{ Definition 4.1.9 }]\!\!\} \\
 & \mathsf{inf}\, C \cup \{c \cdot a \mid c \in \mathsf{fin}\, C,\, a \in \bigcup_{i\in I} A_i,\, c \cdot a \text{ def.}\} \\
= & \quad \{\!\![\text{ set theory }]\!\!\} \\
 & \bigcup_{i\in I} (\mathsf{inf}\, C \cup \{c \cdot a \mid c \in \mathsf{fin}\, C,\, a \in A_i,\, c \cdot a \text{ def.}\}) \\
= & \quad \{\!\![\text{ Definition 4.1.9 }]\!\!\} \\
 & \bigcup_{i\in I} (C \cdot A_i)
\end{array}
$$

  Axiom (3.9) (left annihilation):
  By Definition 4.1.9, set theory and $\mathsf{fin}\,\emptyset = \mathsf{inf}\,\emptyset = \emptyset$,

$$\emptyset \cdot A = \mathsf{inf}\,\emptyset \cup \{c \cdot a \mid c \in \mathsf{fin}\,\emptyset,\, a \in A,\, c \cdot a \text{ def.}\} = \emptyset\ .$$

  Furthermore, it is easy to see that $A \cdot \emptyset \neq \emptyset$ if A contains at least one trajectory with infinite duration ($\mathsf{inf}\, a \neq \emptyset$).

Next we show that PRO forms a Boolean quantale. We have already proved that multiplication is universally disjunctive (positively disjunctive and $A \cdot \emptyset = \emptyset$). Since PRO is a set-based construction and the natural order coincides with the set-theoretic order, PRO forms a complete lattice. The complement in PRO coincides with the standard complement of sets.

2. This is again due to the fact that PRO is set based.

3. The properties of chop are straightforward by its definition. □

Theorem 4.4.7 Let A be purely finite process and let $H : \mathsf{PRO} \to \mathsf{PRO}$ be a function defined by $H(X) =_{df} A \cdot X$.

1. Let X be expanded by H, i.e., assume $X \subseteq H(X)$. Then for every $\xi \in X$ there is a guarded string $w \in \mathsf{GS}$ such that the set of prefixes $\mathsf{pre}(w) \subseteq \mathsf{PRE}(A)$ and $\tau \sqsubseteq \xi$ for all $\tau \in \mathsf{pre}(w)$.
2. $A^\omega = \{\xi \in \mathsf{TRA} \mid \exists\, w \in \mathsf{inf\,GS} : \mathsf{pre}(w) \subseteq \mathsf{PRE}(A), \forall\, \tau \in \mathsf{pre}(w) : \tau \sqsubseteq \xi\}$.

Proof.

1. Consider $\xi \in X$. We inductively construct a sequences of prefixes of ξ. Since $X \subseteq A \cdot X$, there are $\tau_0 \in A$ and $\xi_0 \in X$ with $\xi = \tau_0 \cdot \xi_0$. Since $\xi_0 \in X$, we can again do the same step and define trajectories $\tau_1 \in A$ and $\xi_1 \in X$ such that $\xi = \tau_0 \cdot \xi_0 = \tau_0 \cdot \tau_1 \cdot \xi_1$. In general for $\xi_i \in A \cdot H$ there are trajectories $\tau_{i+1} \in A$ and $\xi_{i+1} \in X$ with $\xi_i = \tau_{i+1} \cdot \xi_{i+1}$. By construction $\prod_{i=1}^{n} \tau_i \sqsubseteq \xi$. Now we choose w as the supremum of all these trajectories lifted to guarded strings, i.e, $\sup\{w \mid w \in \prod_{i=1}^{n} \iota(\tau_i),\, n \in \mathbf{N}\}$ and we are done.

2. As a prerequisite we set $\mathrm{OM}(A) =_{df} \{\xi \in \mathsf{TRA} \mid \exists\, w \in \mathsf{inf\,GS} : \mathsf{pre}(w) \subseteq \mathsf{PRE}(A),\ \forall\, \tau \in \mathsf{pre}(w) : \tau \sqsubseteq \xi\}$ and observe that finite trajectories τ are left cancellative w.r.t. composition, i.e., satisfy $\tau \cdot \rho = \tau \cdot \sigma \Rightarrow \rho = \sigma$, provided $\tau \cdot \rho$ and $\tau \cdot \sigma$ are defined. By omega unfold every guarded string $w \in (\iota(A))^\omega$ has a prefix $w_0 \in \iota(A)$ with $w_0 \sqsubseteq w$.

   Now we show that $\mathrm{OM}(A)$ is expanded by H. Consider an arbitrary $\xi \in \mathrm{OM}(A)$. By definition there is a $w \in \mathsf{GS}$ with $\mathsf{pre}(w) \subseteq \mathsf{PRE}(A)$ and $\tau \sqsubseteq \xi$ for all $\tau \in \mathsf{pre}(w)$. By this and the above remark we know that there is a $w_0 \sqsubseteq w$ with $\phi(w_0) \in \mathsf{pre}(w)$ and $\phi(w_0) \in \phi(\iota(A)) = A$. Then by finiteness of $\phi(w_0)$ and the above cancellation property, there is a unique τ_1 with $\xi = \phi(w_0) \cdot \tau_1$. Hence $\mathrm{OM}(A) \subseteq A \cdot \mathrm{OM}(A)$.

   Together with Part (1) this means that $\mathrm{OM}(A)$ is the greatest expanded element of H and hence its greatest fixpoint. Now the claim follows by Lemma 3.2.9(2). □

Equation (4.4) For an arbitrary set of guarded strings $L \in \mathsf{GS}(\mathsf{test}(\mathsf{PRO}), \mathsf{fin}\,(\mathsf{TRA}))$ we have $\phi(L^\omega) = (\phi(L))^\dagger$ if $L \cap \mathsf{test}(\mathsf{PRO}) = \emptyset$.

Proof. By definition of dagger, the claim is equivalent to

$$\phi\Big(\bigcup_{w \in L^\omega} w\Big) = \bigcup_{v \in (\iota(\phi(L)))^\omega} \widehat{\sup}(\mathsf{pre}(v)) \ .$$

Moreover it is straightforward to see that the homomorphism ϕ is disjunctive. This transforms the left hand side to $\bigcup_{w \in L^\omega} \phi(w)$. From the assumption $L \cap \mathsf{test}(\mathsf{PRO}) = \emptyset$ we get that all guarded string $w \in L^\omega$ and $v \in (\iota(\phi(L)))^\omega$ have infinite length, i.e., $|w| = |v| = \infty$. Hence the claim becomes

$$\bigcup_{w \in L^\omega} \phi(w) = \bigcup_{v \in (\iota(\phi(L)))^\omega} \phi(v) \ .$$

Therefore it is sufficient to show that for all $w \in L^\omega$ there is a $v \in (\iota(\phi(L)))^\omega$ with $\phi(w) = \phi(v)$ and vice versa. In Section 4.4 we have shown that an infinite guarded string can be determined by a set of increasing prefixes. Therefore for an infinite guarded string $w \in L^\omega$ there are guarded strings $w_i \in L$ ($i \in \mathbb{N}$) with $\prod_{i=0}^n w_i \sqsubseteq w$ and $\prod_{i=0}^n w_i \in L^i$. This construction is analogue to the one done in the proof of Theorem 4.4.7. By this we have defined an increasing chain of prefixes. It is straightforward that the limit equals w. We now define $v_i =_{df} \iota(\phi(w_i))$ and its limit $v =_{df} \sup\{\prod_{i=0}^n v_i \mid n \in \mathbb{N}\}$, which verifies the first part of the above claim. Similarly, one can show that for all $v \in (\iota(\phi(L)))^\omega$ there is a $w \in L^\omega$ with $\phi(w) = \phi(v)$. □

Theorem 4.4.14 Let H be as in Theorem 4.4.7.

1. $A^\dagger$ is a fixpoint of H.
2. Let X be expanded by H, i.e., assume $X \subseteq H(X)$. Then every $\tau \in X$ has a prefix in $A^\dagger$.
3. $A^\omega = A^\dagger \cdot \top$.

Proof.

1. The proof is a straightforward calculation. To increase readability we write ιA instead of $\iota(A)$. We first observe again that $\phi(\iota A) = A$. From this we get by definition of dagger, property of ϕ and omega unfold

$$A \cdot A^\dagger = A \cdot \phi((\iota A)^\omega) = \phi(\iota A) \cdot \phi((\iota A)^\omega) = \phi(\iota A \cdot (\iota A)^\omega) = \phi((\iota A)^\omega) = A^\dagger \ .$$

2. Consider an arbitrary $\xi \in X \subseteq A \cdot X$. By Theorem 4.4.7 there is a set $\mathsf{pre}(w)$ with $\tau \sqsubseteq \xi$ for all $\tau \in \mathsf{pre}(w)$. By definition of $\widehat{\sup}(\mathsf{pre}(w))$ we have for all $\sigma \in \widehat{\sup}(\mathsf{pre}(w))$ and all $\tau \in \mathsf{pre}(w)$ $\tau \sqsubseteq \sigma$ If $\widehat{\sup}(\mathsf{pre}(w))$ contains only one single trajectory (no proper Zeno effect occurs) $\sigma_0 =_{df} \widehat{\sup}(\mathsf{pre}(w)) \sqsubseteq \xi$. In the case of Zeno effects there is a trajectory $\sigma_0 \in \widehat{\sup}(\mathsf{pre}(w))$ with $\sigma_0 \sqsubseteq \xi$. σ is the "limit" of all $\tau \in \mathsf{pre}(w)$ that coincides with ξ at time d; hence $\sigma \sqsubseteq \xi$. Since $\sigma_0 \in \widehat{\sup}(\mathsf{pre}(w) \subseteq A^\dagger$, we are done.

3. The claim directly follows from Corollary 4.4.11 and definition of dagger. □

Lemma 4.5.2 Assume a Boolean weak semiring (Kleene algebra).

1. The following properties are equivalent.
   (a) Element $c \in S$ is submodular.
   (b) $(c \sqcap \mathsf{F}) \cdot \overline{c} + \overline{c} \cdot \top \leq \overline{c}$.
   (c) $\mathsf{F} \cdot \overline{c} \cdot \top \leq \overline{c}$.

   In particular, 1 is submodular iff $\overline{1} \cdot \overline{1} \leq \overline{1}$.

2. Element $c \in S$ is modular iff it is submodular and *transitive*, i.e., satisfies $c \cdot c \leq c$. In particular, 1 is modular iff $\overline{1} \cdot \overline{1} \leq \overline{1}$.

3. Submodular elements are closed under addition and meet, hence form a lattice.

4. If c is modular then for all a we have $c \sqcap a^+ = (c \sqcap a)^+$ and $c \sqcap a^* = (c \sqcap a)^+ + (c \sqcap 1)$, with $b^+ =_{df} b \cdot b^*$.

5. If c is modular then $d \leq c$ is submodular iff $(c \sqcap \mathsf{F}) \cdot (c \sqcap \overline{d}) \cdot c \leq c \sqcap \overline{d}$.

Proof.

1. ((a)⇒(b)) The claim is equivalent to $c \sqcap ((c \sqcap \mathsf{F}) \cdot \overline{c} + \overline{c} \cdot \top) \leq 0$ by shunting (Lemma 3.2.3). Then by Boolean algebra, submodularity applied twice, Boolean algebra again, left annihilation and $c \sqcap \mathsf{F} \leq \mathsf{F}$,

$$\begin{array}{ll} & c \sqcap ((c \sqcap \mathsf{F}) \cdot \overline{c} + \overline{c} \cdot \top) \\ = & (c \sqcap (c \sqcap \mathsf{F}) \cdot \overline{c}) + (c \sqcap \overline{c} \cdot \top) \\ \leq & (c \sqcap c \sqcap \mathsf{F}) \cdot (c \sqcap \overline{c}) + (c \sqcap \overline{c}) \cdot (c \sqcap \top) \\ = & (c \sqcap \mathsf{F}) \cdot 0 + 0 \cdot (c \sqcap \top) \; = \; 0 \; . \end{array}$$

   ((b)⇒(c)) By Boolean algebra, distributivity, (b) and isotony,

$$\begin{aligned} \mathsf{F} \cdot \overline{c} \cdot \top = (c \sqcap \mathsf{F} + \overline{c} \sqcap \mathsf{F}) \cdot \overline{c} \cdot \top &= (c \sqcap \mathsf{F}) \cdot \overline{c} \cdot \top + (\overline{c} \sqcap \mathsf{F}) \cdot \overline{c} \cdot \top \\ &\leq \overline{c} \cdot \top + \overline{c} \cdot \top \leq \overline{c} \; . \end{aligned}$$

((c)⇒(a)) Consider first a product $a \cdot b$ with purely finite a, i.e., with $a \leq \mathsf{F}$. By Boolean algebra and distributivity,

$$a \cdot b \;=\; (c \sqcap a) \cdot (c \sqcap b) + (c \sqcap a) \cdot (\overline{c} \sqcap b) + (\overline{c} \sqcap a) \cdot b \;.$$

By $a \leq \mathsf{F}$ and the assumption about c, we have $\mathsf{F} \cdot \overline{c} \leq \overline{c}$ and $\overline{c} \cdot \top \leq \overline{c}$, so that the last two summands are $\leq \overline{c}$ by isotony. Hence,

$$c \sqcap a \cdot b \;=\; c \sqcap (c \sqcap a) \cdot (c \sqcap b) \leq (c \sqcap a) \cdot (c \sqcap b) \;.$$

For arbitrary a we calculate, using $\mathsf{fin}/\mathsf{inf}$ decomposition, Boolean algebra and the claim for $\mathsf{fin}\, a \leq \mathsf{F}$,

$$\begin{array}{ll} & c \sqcap a \cdot b \\ = & c \sqcap (\mathsf{inf}\, a + \mathsf{fin}\, a \cdot b) \\ = & (c \sqcap \mathsf{inf}\, a) + (c \sqcap (\mathsf{fin}\, a \cdot b)) \\ \leq & (c \sqcap \mathsf{inf}\, a) + (c \sqcap \mathsf{fin}\, a) \cdot (c \sqcap b) \\ = & \mathsf{inf}\, (c \sqcap a) + \mathsf{fin}\, (c \sqcap a) \cdot (c \sqcap b) \\ = & (c \sqcap a) \cdot (c \sqcap b) \;. \end{array}$$

Finally, for $c = 1$ the left hand side of Formula (b) spells out to $(1 \sqcap \mathsf{F}) \cdot \overline{1} + \overline{1} \cdot \top = 1 \cdot \overline{1} + \overline{1} \cdot \overline{1} + \overline{1} \cdot 1 = \overline{1} + \overline{1} \cdot \overline{1}$, which shows the claim.

2. (⇒) We only need to show transitivity of c, which holds by

$$c \sqcap c \cdot c = (c \sqcap c) \cdot (c \sqcap c) = c \cdot c \;.$$

(⇐) By isotony, $(c \sqcap a) \cdot (c \sqcap b) \leq a \cdot b$ and $(c \sqcap a) \cdot (c \sqcap b) \leq c \cdot c \leq c$, which shows $(c \sqcap a) \cdot (c \sqcap b) \leq c \sqcap a \cdot b$. The reverse inequation holds by submodularity of c. The assertion about 1 follows, since 1 is transitive.

3. We assume two submodular elements c and d. By Boolean algebra, submodularity, isotony and idempotence we have

$$\begin{array}{ll} & (c + d) \sqcap (a \cdot b) \\ = & c \sqcap (a \cdot b) + d \sqcap (a \cdot b) \\ \leq & (c \sqcap a) \cdot (c \sqcap b) + (d \sqcap a) \cdot (d \sqcap b) \\ \leq & ((c + d) \sqcap a) \cdot ((c + d) \sqcap b) + ((c + d) \sqcap a) \cdot ((c + d) \sqcap b) \\ = & ((c + d) \sqcap a) \cdot ((c + d) \sqcap b) \;. \end{array}$$

This shows that submolar elements are closed under $+$. To show the closedness of $\sqcap$ we use associativity, submodularity (twice) and associativity again.

$$
\begin{array}{ll}
 & (c \sqcap d) \sqcap (a \cdot b) \\
= & c \sqcap (d \sqcap (a \cdot b)) \\
\leq & c \sqcap (d \sqcap a) \cdot (d \sqcap b) \\
\leq & (c \sqcap (d \sqcap a)) \cdot (c \sqcap (d \sqcap b)) \\
= & ((c \sqcap d) \sqcap a) \cdot ((c \sqcap d) \sqcap b)
\end{array}
$$

4. First, we note that by right-distributivity $a^* \cdot a = a \cdot a^*$. Moreover if a is transitive we get $a^+ = a$ (cf. Lemma3.4.2).
($\geq$) Isotony and $c \cdot c \leq c$ show $(c \sqcap a)^+ \leq a^+$ and $(c \sqcap a)^+ \leq c^+ = c$.
($\leq$) By Lemma 3.2.3, definition of $^+$, the above remark, star induction (3.13), distributivity and join splitting we have

$$
\begin{array}{ll}
 & c \sqcap a^+ \leq (c \sqcap a)^+ \\
\Leftrightarrow & a \cdot a^* \leq \overline{c} + (c \sqcap a)^+ \\
\Leftrightarrow & a^* \cdot a \leq \overline{c} + (c \sqcap a)^+ \\
\Leftarrow & a + a \cdot (\overline{c} + (c \sqcap a)^+) \leq \overline{c} + (c \sqcap a)^+ \\
\Leftrightarrow & c \sqcap (a + a \cdot (\overline{c} + (c \sqcap a)^+)) \leq (c \sqcap a)^+ \\
\Leftrightarrow & c \sqcap a \leq (c \sqcap a)^+ \wedge c \sqcap a \cdot \overline{c} \leq (c \sqcap a)^+ \wedge c \sqcap a \cdot (c \sqcap a)^+ \leq (c \sqcap a)^+
\end{array}
$$

The first conjunct holds by neutrality, isotony and $1 \leq (c \sqcap a)^*$. For the second one we have, by modularity of c,

$$c \sqcap a \cdot \overline{c} = (c \sqcap a) \cdot (c \sqcap \overline{c}) = (c \sqcap a) \cdot 0 \leq c \sqcap a \leq (c \sqcap a)^+ .$$

The third conjunct is shown, using again modularity, by

$$c \sqcap a \cdot (c \sqcap a)^+ = (c \sqcap a) \cdot (c \sqcap (c \sqcap a)^+) \leq (c \sqcap a) \cdot (c \sqcap a)^+ \leq (c \sqcap a)^+ .$$

The equation for * is then immediate from $a^* = a^+ + 1$, the equation for $^+$ and distributivity of $\sqcap$.

5. Assume $d \leq c$. Then by Boolean algebra $\overline{d} = \overline{c} + c \sqcap \overline{d}$. By this, shunting (Lemma 3.2.3), modularity (twice) and Boolean algebra, we have

$$
\begin{array}{ll}
 & \mathsf{F} \cdot \overline{d} \cdot \top \leq \overline{d} \\
\Leftrightarrow & \mathsf{F} \cdot \overline{d} \cdot \top \leq \overline{c} + c \sqcap \overline{d} \\
\Leftrightarrow & c \sqcap \mathsf{F} \cdot \overline{d} \cdot \top \leq c \sqcap \overline{d} \\
\Leftrightarrow & (c \sqcap \mathsf{F}) \cdot (c \sqcap \overline{d}) \cdot (c \sqcap \top) \leq c \sqcap \overline{d} \\
\Leftrightarrow & (c \sqcap \mathsf{F}) \cdot (c \sqcap \overline{d}) \cdot c \leq c \sqcap \overline{d} .
\end{array}
$$

□

Lemma 4.5.13 Assume a Boolean weak semiring S that is safety-closed.

1. All boxes are modular.
2. All boxes are multiplicatively idempotent, i.e., $(\Box p) \cdot (\Box p) = \Box p$.
3. $\Box p \sqcap a^+ = (\Box p \sqcap a)^+$ and $\Box p \sqcap a^* = (\Box p \sqcap a)^+ + (\Box p \sqcap 1)$.
4. $\diamondsuit p \sqcap a \cdot b = (\diamondsuit p \sqcap a) \cdot b + a^\frown(\diamondsuit p \sqcap b)$.

Proof.

1. Immediate from Lemma 4.5.9, Lemma 4.5.2(2) and safety-closedness, i.e., transitivity of boxes.
2. This is a consequence of Part (1), since $\Box p = \Box p \sqcap \top = \Box p \sqcap \top \cdot \top = (\Box p \sqcap \top) \cdot (\Box p \sqcap \top) = \Box p \cdot \Box p$.
3. Immediate from Part (1) and Lemma 4.5.2(4).
4. We show the claim for purely finite a. For purely infinite a the proof is straightforward since $a \cdot b = a$. For general a the proof proceeds by splitting a into its purely finite and purely infinite part. Set $d =_{df} \diamondsuit p$ and $s =_{df} \overline{d} = \Box\neg p$. By Boolean algebra and distributivity,

$$d \sqcap a \cdot b = d \sqcap (d \sqcap a) \cdot b + d \sqcap (s \sqcap a) \cdot (d \sqcap b) + d \sqcap (s \sqcap a) \cdot (s \sqcap b) \ .$$

The first of these summands is below $(d \sqcap a) \cdot b$, the second one is below $a \cdot (d \sqcap b)$ and the third one is 0 by Part (1) and $d \sqcap s = 0$. Hence, the sum is below $(d \sqcap a) \cdot b + a \cdot (d \sqcap b)$.
The converse inequation holds by $d \cdot b \leq d$, $a \leq \mathsf{F}$, $\mathsf{F} \cdot d \leq d$ and isotony. □

Theorem 4.5.20 Assume a Boolean weak and locally linear quantale S. Then for all $a, b \in S$ and $p, q \in \mathsf{test}(S)$ the following properties hold.

1. $a \cdot b \sqcap \mathbb{F}p \cdot \Box q = (a \sqcap \mathbb{F}p) \cdot (b \sqcap \Box q) + (a \sqcap \mathbb{F}p) \cdot (b \sqcap \mathbb{F}p \cdot \Box q) + (a \sqcap \mathbb{F}p \cdot \Box q) \cdot (b \sqcap \Box q)$.
2. $a \cdot b \sqcap \mathbb{N}p = (a \sqcap \mathbb{N}p) + (a \sqcap \mathbb{F}p) \cdot (b \sqcap \mathbb{N}p) = (a \sqcap \Box p) \cdot (b \sqcap \mathbb{N}p)$.
3. $a \cdot b \sqcap \Box p \cdot \Box q = (a \sqcap \Box p) \cdot (b \sqcap \Box q) + (a \sqcap \Box p) \cdot (b \sqcap \Box p \cdot \Box q) + (a \sqcap \Box p \cdot \Box q) \cdot (b \sqcap \Box q)$.
4. If additionally $p \leq \Box p$ holds, the summand $(a \sqcap \mathbb{F}p) \cdot (b \sqcap \Box q)$ can be omitted from the right hand sides of Parts (1) and (3).

Proof.

1. We show the claim first for purely finite a, i.e., for $a \leq \mathsf{F}$. The inequation $(\geq)$ holds by isotony and safety-closedness. For $(\leq)$ we calculate

$$
\begin{array}{ll}
 & a \cdot b \sqcap \mathbb{F}p \cdot \Box q \\
= & \quad \{\![\text{ idempotence }]\!\} \\
 & a \cdot b \sqcap \mathbb{F}p \cdot \Box q \sqcap \mathbb{F}p \cdot \Box q \\
\leq & \quad \{\![\text{ by Lemma B.1.2(5) }]\!\} \\
 & ((a \sqcap \mathbb{F}p) \cdot b + (a \sqcap \mathbb{F}p \cdot \Box q) \cdot b) \sqcap \mathbb{F}p \cdot \Box q \\
= & \quad \{\![\text{ idempotence and distributivity }]\!\} \\
 & [((a \sqcap \mathbb{F}p) \cdot b \sqcap \mathbb{F}p \cdot \Box q) + ((a \sqcap \mathbb{F}p \cdot \Box q) \cdot b \sqcap \mathbb{F}p \cdot \Box q)] \sqcap \mathbb{F}p \cdot \Box q \\
\leq & \quad \{\![\text{ by Lemma B.1.2(6) (twice) }]\!\} \\
 & [(a \sqcap \mathbb{F}p) \cdot (b \sqcap \mathbb{F}p \cdot \Box q) + (a \sqcap \mathbb{F}p) \cdot (b \sqcap \Box q) + \\
 & \ (a \sqcap \mathbb{F}p \cdot \Box q) \cdot (b \sqcap \mathbb{F}p \cdot \Box q) + (a \sqcap \mathbb{F}p \cdot \Box q) \cdot (b \sqcap \Box q)] \sqcap \mathbb{F}p \cdot \Box q \\
\leq & \quad \{\![\text{ distributivity and omitting meets }]\!\} \\
 & (a \sqcap \mathbb{F}p) \cdot (b \sqcap \mathbb{F}p \cdot \Box q) + (a \sqcap \mathbb{F}p) \cdot (b \sqcap \Box q) + \\
 & \ (a \sqcap \mathbb{F}p \cdot \Box q) \cdot (b \sqcap \mathbb{F}p \cdot \Box q) + ((a \sqcap \mathbb{F}p \cdot \Box q) \cdot (b \sqcap \Box q) \sqcap \mathbb{F}p \cdot \Box q) \ .
\end{array}
$$

It remains to show that the summand $(a \sqcap \mathbb{F}p \cdot \Box q) \cdot (b \sqcap \Box q) \sqcap \mathbb{F}p \cdot \Box p$ is below the sum of the other summands. We first deal with the left conjoint:

$$
\begin{array}{ll}
 & (a \sqcap \mathbb{F}p \cdot \Box q) \cdot (b \sqcap \mathbb{F}p \cdot \Box q) \\
= & \quad \{\![\text{ Boolean algebra }]\!\} \\
 & (a \sqcap \mathbb{F}p \cdot \Box q \sqcap \mathbb{F}p + a \sqcap \mathbb{F}p \cdot \Box q \sqcap \overline{\mathbb{F}p}) \cdot \\
 & (b \sqcap \mathbb{F}p \cdot \Box q \sqcap \Box q + b \sqcap \mathbb{F}p \cdot \Box q \sqcap \overline{\Box q}) \\
= & \quad \{\![\text{ distributivity }]\!\} \\
 & (a \sqcap \mathbb{F}p \cdot \Box q \sqcap \mathbb{F}p) \cdot (b \sqcap \mathbb{F}p \cdot \Box q \sqcap \Box q) + \\
 & (a \sqcap \mathbb{F}p \cdot \Box q \sqcap \mathbb{F}p) \cdot (b \sqcap \mathbb{F}p \cdot \Box q \sqcap \overline{\Box q}) + \\
 & (a \sqcap \mathbb{F}p \cdot \Box q \sqcap \overline{\mathbb{F}p}) \cdot (b \sqcap \mathbb{F}p \cdot \Box q \sqcap \Box q) + \\
 & (a \sqcap \mathbb{F}p \cdot \Box q \sqcap \overline{\mathbb{F}p}) \cdot (b \sqcap \mathbb{F}p \cdot \Box q \sqcap \overline{\Box q}) \\
\leq & \quad \{\![\text{ omitting meets and definition of box }]\!\} \\
 & (a \sqcap \mathbb{F}p) \cdot (b \sqcap \mathbb{F}p \cdot \Box q) + (a \sqcap \mathbb{F}p \cdot \Box q) \cdot (b \sqcap \Box q) + \\
 & (a \sqcap \mathbb{F}p \cdot \Box q \sqcap \overline{\mathbb{F}p}) \cdot (b \sqcap \mathbb{F}p \cdot \Box q \sqcap \Diamond\!\!\!\!\!\!\bigcirc \neg q) \ .
\end{array}
$$

Hence, by distributivity and omitting meets,

$$
\begin{array}{ll}
 & (a \sqcap \mathbb{F}p \cdot \Box q) \cdot (b \sqcap \Box q) \sqcap \mathbb{F}p \cdot \Box q \\
\leq & (a \sqcap \mathbb{F}p) \cdot (b \sqcap \mathbb{F}p \cdot \Box q) + (a \sqcap \mathbb{F}p \cdot \Box q) \cdot (b \sqcap \Box q) + \\
 & (a \sqcap \mathbb{F}p \cdot \Box q \sqcap \overline{\mathbb{F}p}) \cdot (b \sqcap \mathbb{F}p \cdot \Box q \sqcap \Diamond\!\!\!\!\!\!\bigcirc \neg q) \sqcap \mathbb{F}p \cdot \Box q
\end{array}
$$

and we are done if we can reduce the last summand to 0. For this we calculate

$$
\begin{array}{ll}
 & (a \sqcap \mathbb{F}p \cdot \Box q \sqcap \overline{\mathbb{F}p}) \cdot (b \sqcap \mathbb{F}p \cdot \Box q \sqcap \Diamond\neg q) \sqcap \mathbb{F}p \cdot \Box q \leq 0 \\
\Leftarrow & \quad \{\![\; a \leq \mathsf{F},\ \mathsf{F} \sqcap \overline{\mathbb{F}p} = \mathsf{F} \sqcap \Diamond\neg p \text{ and omitting meets }]\!\} \\
 & (\mathsf{F} \sqcap \Diamond\neg p) \cdot \Diamond\neg q \sqcap \mathbb{F}p \cdot \Box q \leq 0 \\
\Leftrightarrow & \quad \{\![\text{ definition of } \Diamond \text{ and Lemma 3.4.9 }]\!\} \\
 & \mathsf{F} \cdot \neg p \cdot \Diamond\neg q \sqcap \mathbb{F}p \cdot \Box q \leq 0 \\
\Leftrightarrow & \quad \{\![\text{ Lemma 3.2.3 }]\!\} \\
 & \mathsf{F} \cdot \neg p \cdot \Diamond\neg q \leq \overline{\mathbb{F}p \cdot \Box q} \\
\Leftrightarrow & \quad \{\![\text{ exchange rule }]\!\} \\
 & \mathsf{F} \rfloor (\mathbb{F}p \cdot \Box q) \leq \overline{\neg p \cdot \Diamond\neg q} \\
\Leftarrow & \quad \{\![\text{ Lemma B.1.2(3) }]\!\} \\
 & \mathbb{F}p \cdot \Box q + \Box q \leq \overline{\neg p \cdot \Diamond\neg q} \\
\Leftrightarrow & \quad \{\![\text{ supremum }]\!\} \\
 & \mathbb{F}p \cdot \Box q \leq \overline{\neg p \cdot \Diamond\neg q} \ \ \wedge \ \ \Box q \leq \overline{\neg p \cdot \Diamond\neg q} \\
\Leftrightarrow & \quad \{\![\text{ Lemma 3.2.3 }]\!\} \\
 & \neg p \cdot \Diamond\neg q \leq \overline{\mathbb{F}p \cdot \Box q} \ \ \wedge \ \ \neg p \cdot \Diamond\neg q \leq \Diamond\neg q \\
\Leftrightarrow & \quad \{\![\; \neg p \leq 1 \;]\!\} \\
 & \neg p \cdot \Diamond\neg q \leq \overline{\mathbb{F}p \cdot \Box q} \\
\Leftrightarrow & \quad \{\![\text{ shunting (Lemma 3.2.3)) and } \mathbb{F}p = p \cdot \mathbb{F}p \;]\!\} \\
 & \neg p \cdot \Diamond\neg q \sqcap p \cdot \mathbb{F}p \cdot \Box q \leq 0 \\
\Leftrightarrow & \quad \{\![\text{ Lemma 3.5.3 }]\!\} \\
 & p \cdot \neg p \cdot \Diamond\neg q \sqcap \mathbb{F}p \cdot \Box q \leq 0 \\
\Leftrightarrow & \quad \{\![\text{ cancellation law }]\!\} \\
 & 0 \cdot \Diamond\neg q \sqcap \mathbb{F}p \cdot \Box q \leq 0 \\
\Leftrightarrow & \quad \{\![\text{ annihilator }]\!\} \\
 & \mathsf{true} \; .
\end{array}
$$

This finishes the proof for purely finite a. For purely infinite a the claim is trivial, since then $a \sqcap \mathbb{F}p = 0$ and $a \sqcap \mathbb{F}p \cdot \Box q$ is purely infinite again and hence a left zero. For arbitrary a the claim now follows by splitting a into its purely finite and purely infinite parts and using distributivity.

2.

$$
\begin{array}{ll}
 & a \cdot b \sqcap \mathbb{N}p \\
= & \quad \{\![\text{ definition }]\!\} \\
 & a \cdot b \sqcap \mathsf{inf}\, \Box p \\
= & \quad \{\![\text{ Lemma 3.4.13 }]\!\} \\
 & \mathsf{inf}\, (a \cdot b \sqcap \Box p) \\
= & \quad \{\![\text{ by Lemma 4.5.13(1) }]\!\}
\end{array}
$$

$$
\begin{array}{ll}
 & \mathsf{inf}\,(a \sqcap \Box p) \cdot (b \sqcap \Box p) \\
= & \quad \{\![\text{ Lemma 3.4.14(2) }]\!\} \\
 & \mathsf{inf}\,(a \sqcap \Box p) + \mathsf{fin}\,(a \sqcap \Box p) \cdot \mathsf{inf}\,(b \sqcap \Box p) \\
= & \quad \{\![\text{ Lemma 3.4.13 }]\!\} \\
 & (a \sqcap \mathsf{inf}\,\Box p) + (a \sqcap \mathsf{fin}\,\Box p) \cdot (b \sqcap \mathsf{inf}\,\Box p) \\
= & \quad \{\![\text{ definitions }]\!\} \\
 & (a \sqcap \mathbb{N} p) + (a \sqcap \mathbb{F} p) \cdot (b \sqcap \mathbb{N} p) \\
= & \quad \{\![\; a \sqcap \mathbb{N} p \leq \mathsf{N} \;]\!\} \\
 & (a \sqcap \mathbb{N} p) \cdot (b \sqcap \mathbb{N} p) + (a \sqcap \mathbb{F} p) \cdot (b \sqcap \mathbb{N} p) \\
= & \quad \{\![\text{ distributivity }]\!\} \\
 & ((a \sqcap \mathbb{N} p) + (a \sqcap \mathbb{F} p)) \cdot (b \sqcap \mathbb{N} p) \\
= & \quad \{\![\text{ distributivity }]\!\} \\
 & (a \sqcap (\mathbb{N} p + \mathbb{F} p)) \cdot (b \sqcap \mathbb{N} p) \\
= & \quad \{\![\text{ fin/inf decomposition }]\!\} \\
 & (a \sqcap \Box p) \cdot (b \sqcap \mathbb{N} p)
\end{array}
$$

3. First we have

$$
\begin{array}{ll}
 & a \cdot b \sqcap \Box p \cdot \Box q \\
= & \quad \{\![\text{ fin/inf decomposition }]\!\} \\
 & a \cdot b \sqcap (\mathbb{F} p + \mathbb{N} p) \cdot \Box q \\
= & \quad \{\![\text{ distributivity }]\!\} \\
 & a \cdot b \sqcap (\mathbb{F} p \cdot \Box q + \mathbb{N} p \cdot \Box q) \\
= & \quad \{\![\; \mathbb{N} p \cdot \Box q = \mathbb{N} p \;]\!\} \\
 & a \cdot b \sqcap (\mathbb{F} p \cdot \Box q + \mathbb{N} p) \\
= & \quad \{\![\text{ distributivity }]\!\} \\
 & (a \cdot b \sqcap \mathbb{F} p \cdot \Box q) + (a \cdot b \sqcap \mathbb{N} p) \; .
\end{array}
$$

   Then the claim follows from Parts (1) and (3) by straightforward calculation.

4. We have

$$
\begin{array}{ll}
 & (a \sqcap \mathbb{F} p) \cdot (b \sqcap \Box q) \\
= & \quad \{\![\text{ Lemma 4.5.11 }]\!\} \\
 & (a \sqcap \mathbb{F} p \cdot p) \cdot (b \sqcap \Box q) \\
= & \quad \{\![\text{ Corollary 3.5.4 }]\!\} \\
 & (a \sqcap \mathbb{F} p) \cdot p \cdot (b \sqcap \Box q) \\
= & \quad \{\![\text{ Corollary 3.5.4 }]\!\}
\end{array}
$$

$$\begin{array}{ll} & (a \sqcap \mathbb{F} p) \cdot (b \sqcap p \cdot \Box q) \\ \leq & \quad \{\!\![\ p \leq \Box p\ ]\!\!\} \\ & (a \sqcap \mathbb{F} p) \cdot (b \sqcap \mathbb{F} p \cdot \Box q)\ . \end{array}$$

This shows the claim for the right hand side of Part (1) from which the one for Part (3) follows. □

Lemma 5.2.10 Consider a Boolean weak quantale S and $\mathsf{n} \in S$ such that $\mathsf{n} \cdot 0 = 0$. (n must be a purely finite element)

1. $\forall\, b \in S : \mathsf{n} \cdot \overline{b} \leq \overline{\mathsf{n} \cdot b} \Leftrightarrow \forall\, b, c \in S : \mathsf{n} \cdot (b \sqcap c) = \mathsf{n} \cdot b \sqcap \mathsf{n} \cdot c$.
2. $\forall\, b \in S : \overline{\mathsf{n} \cdot b} \leq \mathsf{n} \cdot \overline{b} \Leftrightarrow \mathsf{n} \cdot \top = \top \Leftrightarrow \mathsf{n}^\omega = \top$.

Proof.

1. This property is already shown in [DM01].
   ($\Rightarrow$) It suffices to show ($\geq$), since the reverse inequality follows by isotony. By shunting, the assumption $\mathsf{n} \cdot \overline{b} \leq \overline{\mathsf{n} \cdot b}$, distributivity, Boolean algebra, and lattice algebra:

   $$\begin{array}{c} \mathsf{n} \cdot b \sqcap \mathsf{n} \cdot c \leq \mathsf{n} \cdot (b \sqcap c) \Leftrightarrow \mathsf{n} \cdot b \leq \overline{\mathsf{n} \cdot c} + \mathsf{n} \cdot (b \sqcap c) \Leftarrow \mathsf{n} \cdot b \leq \mathsf{n} \cdot \overline{c} + \mathsf{n} \cdot (b \sqcap c) \\ \Leftrightarrow \mathsf{n} \cdot b \leq \mathsf{n} \cdot (\overline{c} + (b \sqcap c)) \Leftrightarrow \mathsf{n} \cdot b \leq \mathsf{n} \cdot (\overline{c} + b) \Leftrightarrow \mathsf{true}\ . \end{array}$$

   ($\Leftarrow$) We calculate, using the assumption in the third step:

   $$0 = \mathsf{n} \cdot 0 = \mathsf{n} \cdot (b \sqcap \overline{b}) = \mathsf{n} \cdot b \sqcap \mathsf{n} \cdot \overline{b}\ .$$

   Now the claim is immediate by shunting.
2. By shunting, distributivity, complement, greatest element, and $\mathsf{n}^\omega = \nu y\,.\,\mathsf{n} \cdot y$:

   $$\overline{\mathsf{n} \cdot b} \leq \mathsf{n} \cdot \overline{b} \Leftrightarrow \top \leq \mathsf{n} \cdot b + \mathsf{n} \cdot \overline{b} \Leftrightarrow \top \leq \mathsf{n} \cdot (b + \overline{b}) \Leftrightarrow \top \leq \mathsf{n} \cdot \top \Leftrightarrow \top = \mathsf{n} \cdot \top \Leftrightarrow \mathsf{n}^\omega = \top.$$

□

Lemma 5.2.12 Let φ, ψ be state formulas of CTL^* and $p \cdot \top =_{df} [\![\varphi]\!], q \cdot \top =_{df} [\![\psi]\!]$.

1. $[\![\varphi \mathsf{U} \psi]\!] = (p \cdot \mathsf{n})^* \cdot q \cdot \top = ([\![\varphi]\!] \sqcap \mathsf{n})^* \cdot [\![\psi]\!]$.
2. $[\![\mathsf{G}\varphi]\!] = (p \cdot \mathsf{n})^\omega = ([\![\varphi]\!] \sqcap \mathsf{n})^\omega$.
   Hence we have the shunting rule $(p \cdot \mathsf{n})^\omega = \overline{\mathsf{n}^* \cdot \neg p \cdot \top}$.

Proof. For the proof we use knowledge about dual functions and their fixpoints. The *(de Morgan) dual* f^δ of a function $f : S \to S$ over a Boolean quantale is, as usual, defined by $f^\delta(y) =_{df} \overline{f(\overline{y})}$. Then $\mu f = \overline{\nu f^\delta}$ and $\nu f = \overline{\mu f^\delta}$.

1. Using Theorem 5.2.7 and Lemma 3.5.3 we calculate

$$[\![\varphi \mathsf{U} \psi]\!] \;=\; \mu y \,.\, q \cdot \top + (p \cdot \top \sqcap \mathsf{n} \cdot y) \;=\; \mu y \,.\, q \cdot \top + p \cdot \mathsf{n} \cdot y \;,$$

   and the claim follows since $a^* \cdot b = \mu x \,.\, b + x \cdot a$ (cf. Page 39).

2. Since $[\![\mathsf{F}\varphi]\!] = \mu f_p$ where $f_p(y) = p \cdot \top + \mathsf{n} \cdot y$, we have, by Corollary 3.5.4, $[\![\mathsf{G}\varphi]\!] = [\![\neg \mathsf{F} \neg \varphi]\!] = \nu f^{\delta}_{\neg p}$, where, again by Corollary 3.5.4 and by (5.1),

$$f^{\delta}_{\neg p}(y) \;=\; \overline{\neg p \cdot \top + \mathsf{n} \cdot \overline{y}} \;=\; \overline{\neg p \cdot \top} \sqcap \overline{\mathsf{X} \cdot \overline{y}} \;=\; p \cdot \top \sqcap \mathsf{n} \cdot y \;=\; p \cdot \mathsf{n} \cdot y \;.$$

   Hence the claim follows by the definition of ω.

□

Lemma 5.2.15 EX and AX are de Morgan duals, i.e., $[\![\mathsf{AX}\varphi]\!] = [\![\neg \mathsf{EX} \neg \varphi]\!]$. By Lemma 5.2.14 this implies $[\![\mathsf{AX}\varphi]\!] = [\![\mathsf{AXA}\varphi]\!]$. **Proof.** By Theorem 5.2.11(3), the definitions, (5.1), Lemma 3.5.5 and the definitions again, we obtain

$$\begin{aligned} [\![\mathsf{AX}\varphi]\!] \;&=\; \neg \overline{\lceil \overline{[\![\mathsf{X}\varphi]\!]} \cdot \top} \;=\; \neg \overline{\lceil \overline{\mathsf{X} \cdot [\![\varphi]\!]} \cdot \top} \;=\; \neg \overline{\lceil (\mathsf{X} \cdot \overline{[\![\varphi]\!]}) \cdot \top} \\ &=\; \overline{\lceil (\mathsf{X} \cdot \overline{[\![\varphi]\!]}) \cdot \top} \;=\; [\![\neg \mathsf{EX} \neg \varphi]\!] \;. \end{aligned}$$

□

Appendix C

Gate Controller Details

Die Liebe zum Detail verbaut so manchem die
Aussicht auf ein sorgenfreies Leben.
Anonym

This ultimate appendix presents the deferred hybrid automata concerning the railway examples from Section 2.5: We present the strict composed automaton modelling the whole gate controller system. Further the liberal composed automaton is given which models a non-safe system. Last we present the composed hybrid automata for the railway system with a shared single-track way.

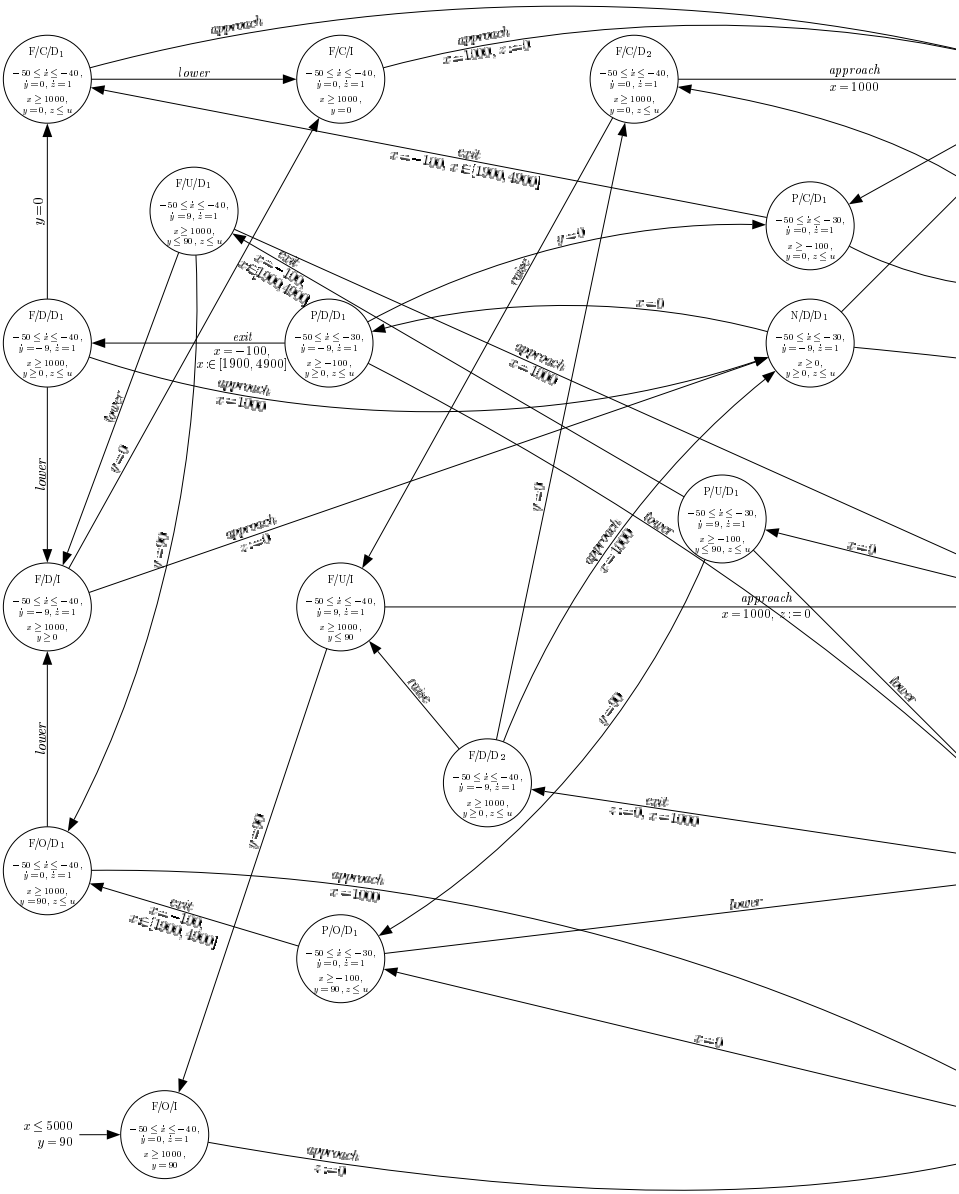


Figure C.1: Strict parallel composed automaton for the gate controller

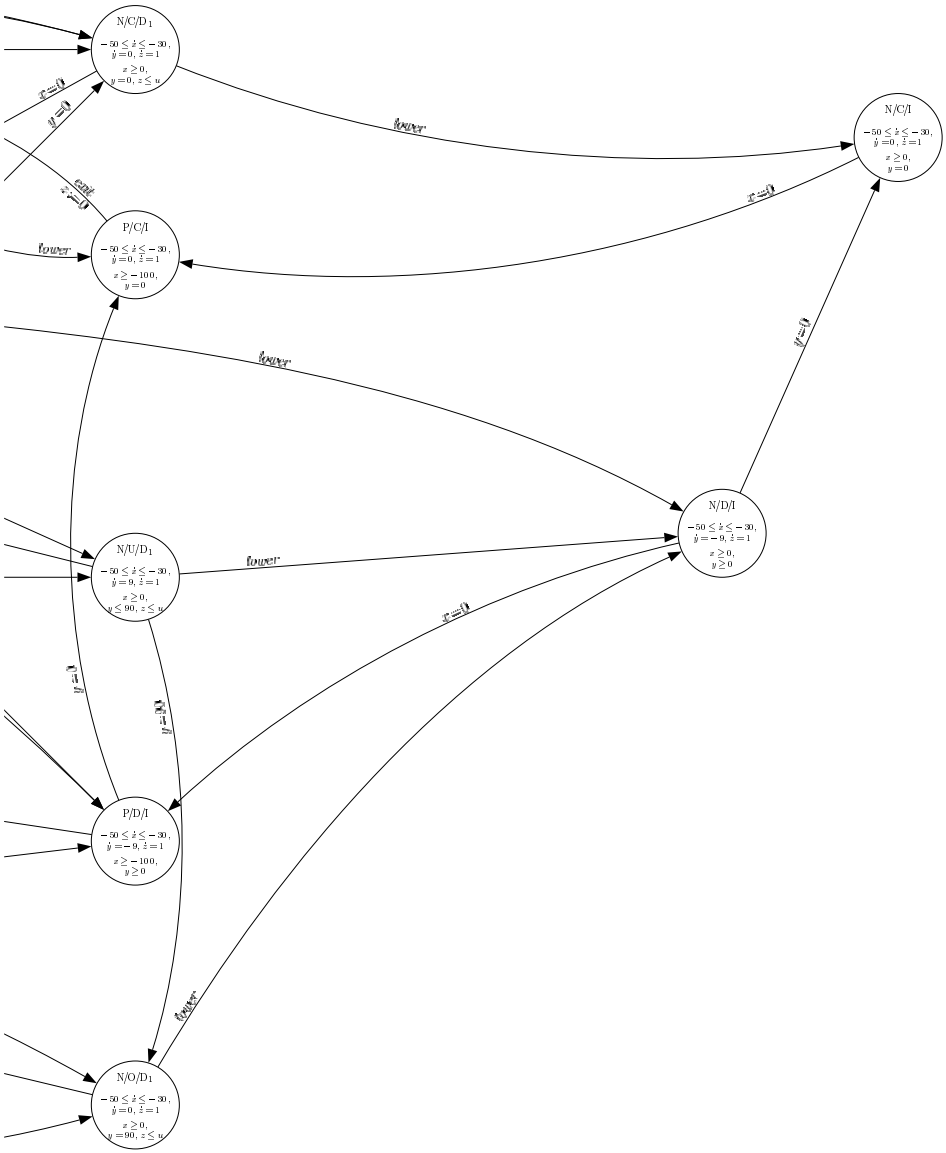


Figure C.1: Strict parallel composed automaton for the gate controller (continued)

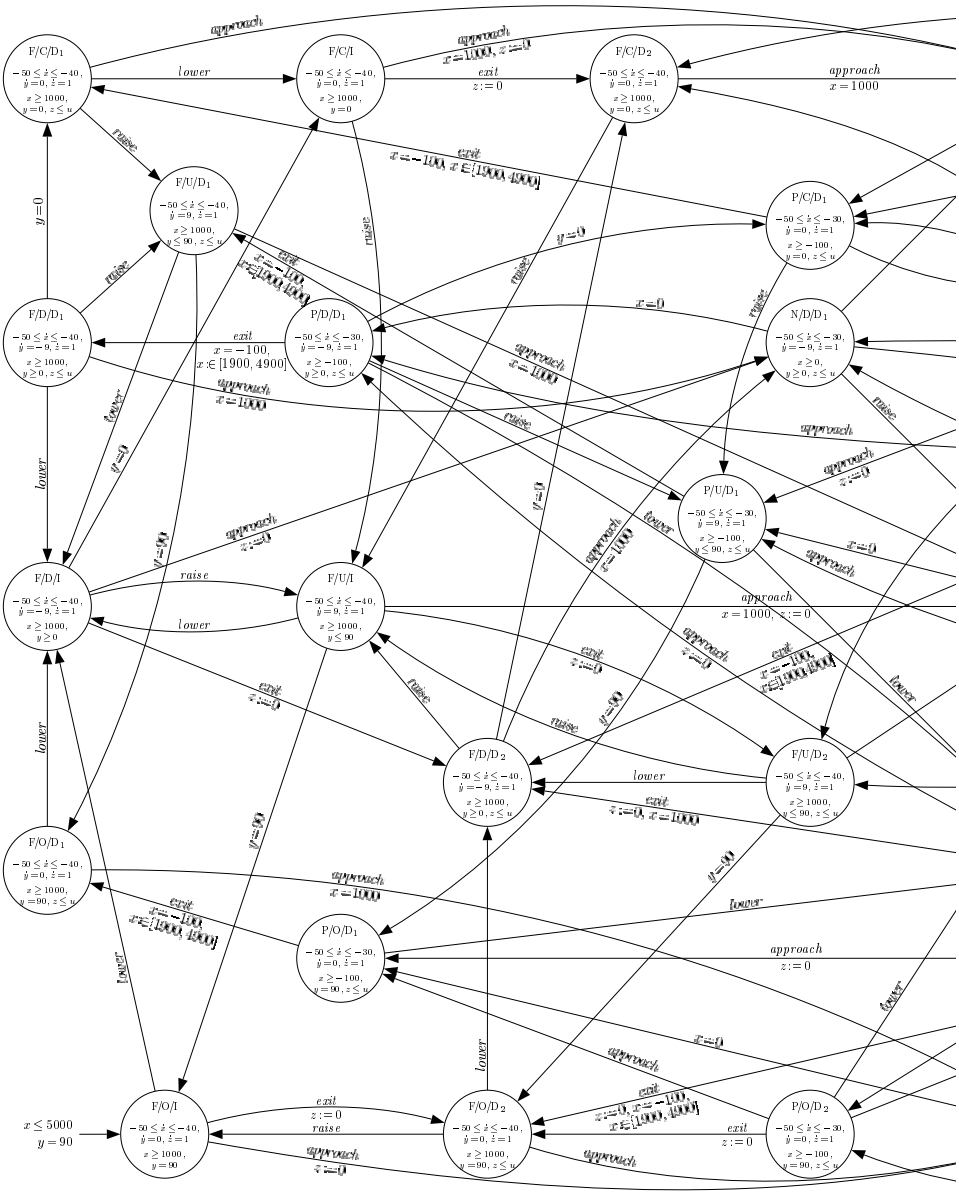


Figure C.2: Liberal parallel composed automaton for the gate controller

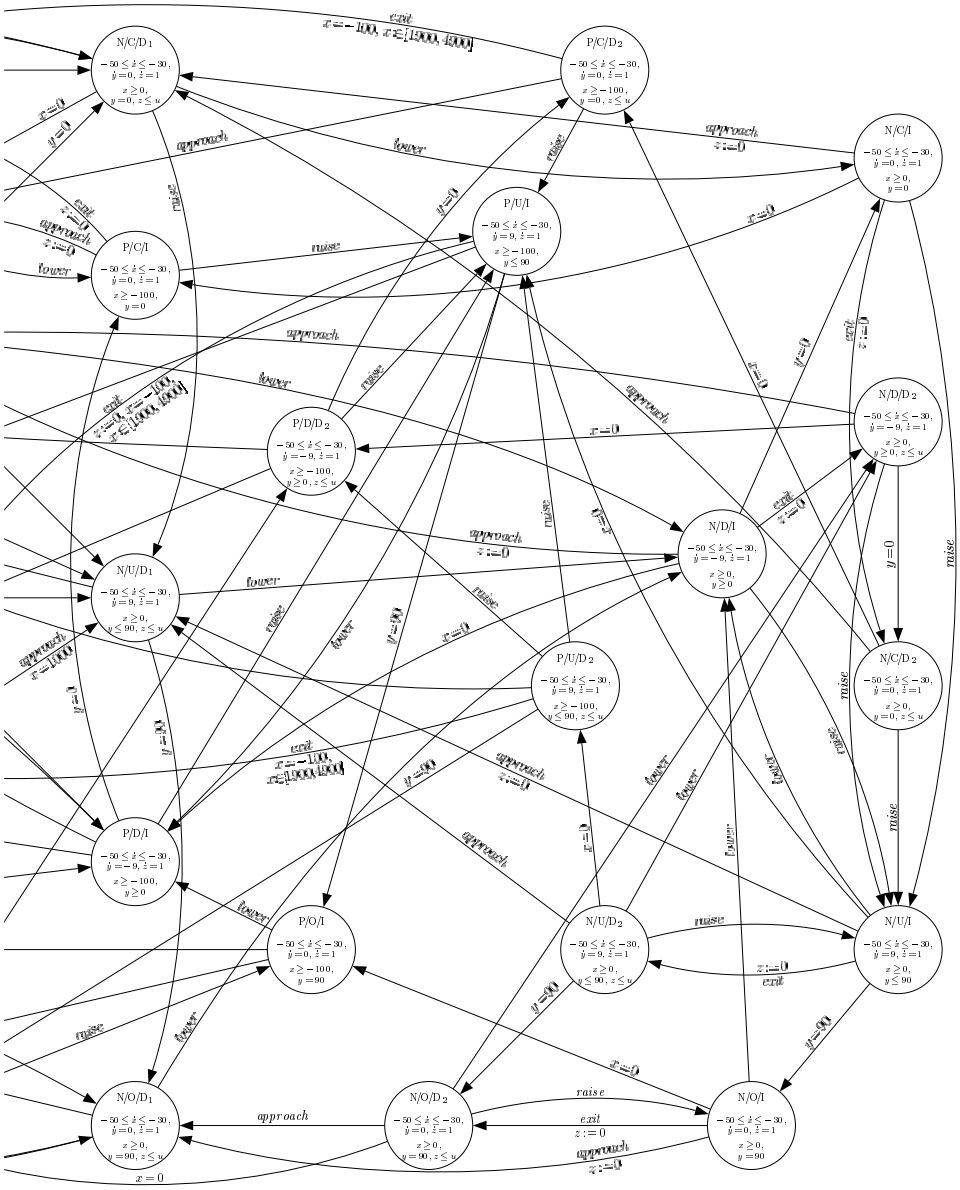


Figure C.2: Liberal parallel composed automaton for the gate controller (continued)

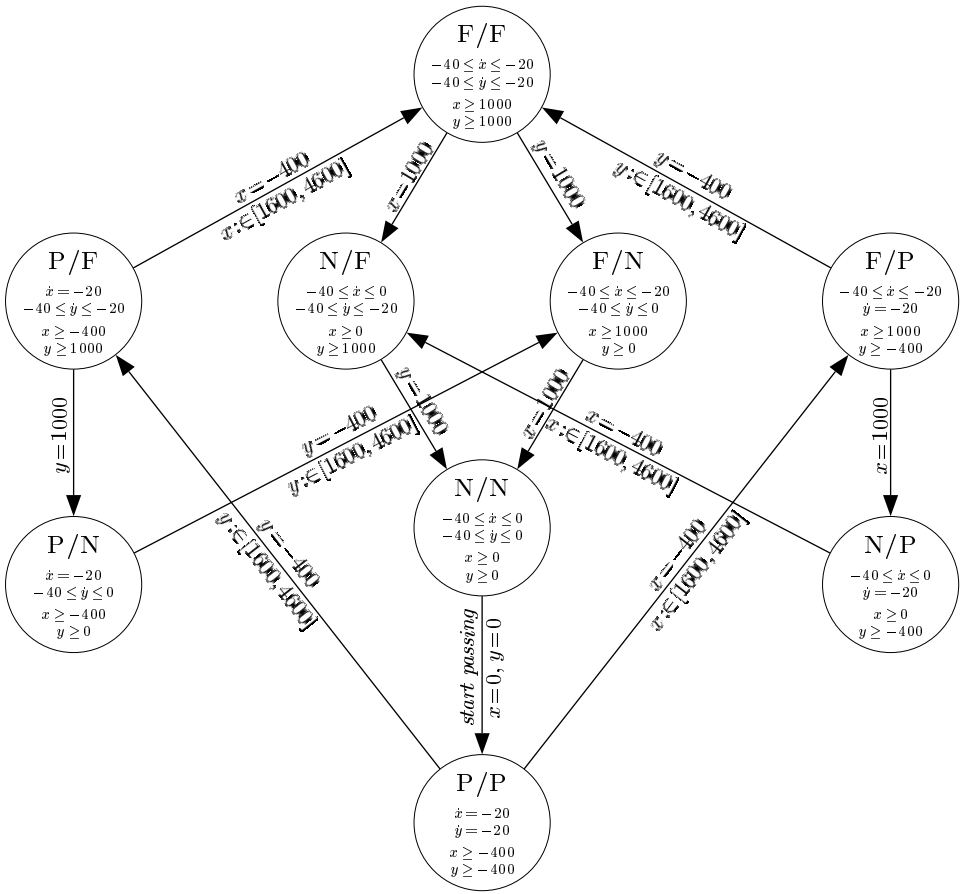


Figure C.3: Strict parallel composed automaton for a shared single-track way

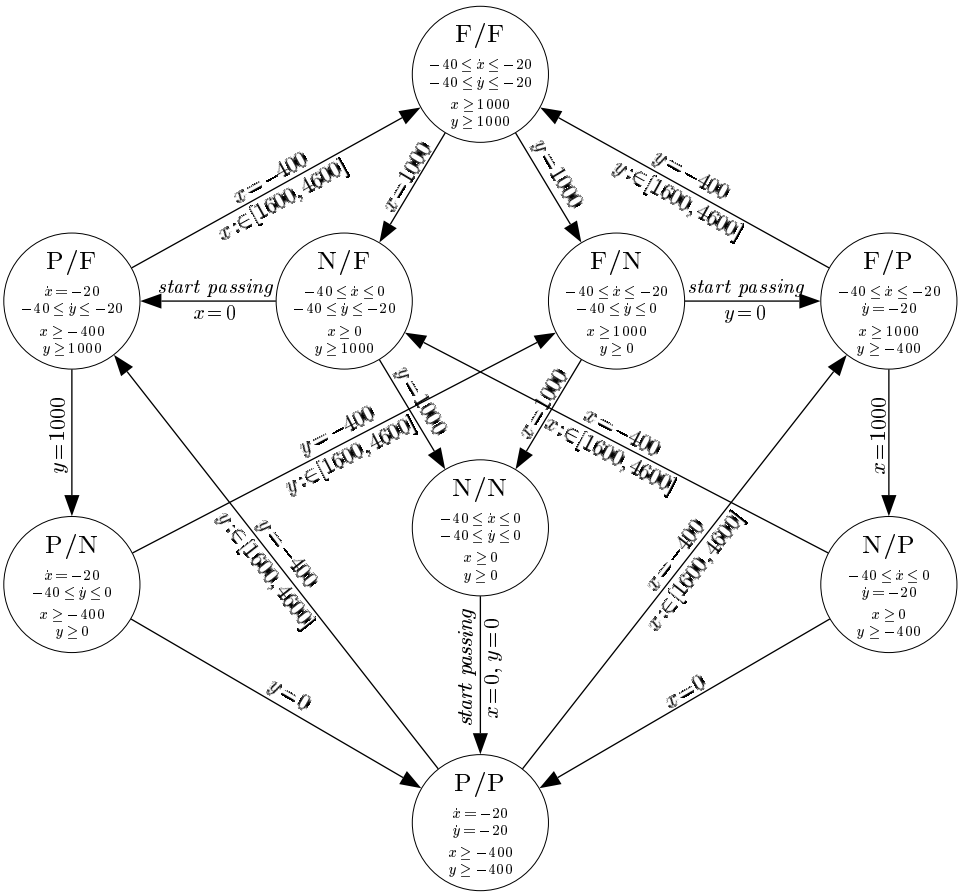


Figure C.4: Liberal parallel composed automaton for a shared single-track way

Bibliography

[AAS05] A. D. Ames, A. Abate, and S. Sastry, *Sufficient conditions for the existence of Zeno behavior*, IEEE Conference on Decision and Control, IEEE Press, 2005.

[ABB+95] C. Aarts, R. Backhouse, E. Boiten, H. Doornbos, N. van Gasteren, R. van Geldrop, P. Hoogendijk, E. Voermans, and J. van der Woude, *Fixed-point calculus*, Information Processing Letters **53** (1995), no. 3, 131–136.

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, *The algorithmic analysis of hybrid systems*, Theoretical Computer Science **138** (1995), no. 1, 3–34.

[ACHH93] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho, *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*, Hybrid Systems (R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, eds.), Lecture Notes in Computer Science, vol. 736, Springer, 1993, pp. 209–229.

[AD94] R. Alur and D. L. Dill, *A theory of timed automata*, Theoretical Computer Science **126** (1994), no. 2, 183–235.

[AGH+00] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee, *Modular specification of hybrid systems in CHARON*, Hybrid Systems: Computation and Control, Lecture Notes in Computer Science, vol. 1790, Springer, 2000, pp. 6–19.

[AH85] J. F. Allen and P. J. Hayes, *A common-sense theory of time*, International Joint Conference on Artificial Intelligence (A. K. Joshi, ed.), vol. 1, 1985, pp. 528–531.

[AHK06] K. Aboul-Hosn and D. Kozen, *KAT-ML: An interactive theorem prover for Kleene algebra with tests*, Journal of Applied Non-Classical Logics **16** (2006), no. 1-2, 9–33.

[AHLP00] R. Alur, T. A. Henzinger, G. Lafferriere, and G. J. Pappas, *Discrete abstractions of hybrid systems*, Proceedings of the IEEE **88** (2000), 971–984.

[All83] J. F. Allen, *Maintaining knowledge about temporal intervals*, ACM Transactions on Computational Logic **26** (1983), no. 11, 832–843.

[AM01] E. Abrahám-Mumm, *Verification of hybrid systems: Formalization and proof rules in PVS*, ICECCS '01: Proceedings of the Seventh International Conference on Engineering of Complex Computer Systems, IEEE Press, 2001, pp. 48–57.

[APLS08] A. Abate, M. Prandini, J. Lygeros, and S. Sastry, *Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems*, Automatica **44** (2008), no. 11, 2724–2734.

[Ard60] D. N. Arden, *Delayed logic and finite state machines*, Theory of Computing Machine Design, University of Michigan Press, 1960, pp. 1–35.

[Arf85] G. Arfgen, *"Separation of variables" and "Separation of variables–ordinary differential equations"*, Mathematical Methods for Physicists, ch. 2.6 and 8.3, pp. 111–117 and 448–451, Academic Press, 1985.

[ASIM02] E. Aaron, H. Sun, F. Ivancic, and D. Metaxas, *A hybrid dynamical systems approach to intelligent low-level navigation*, Computer Animation (2002), 154–163.

[Bac02] R. Backhouse, *Galois connections and fixed point calculus.*, Algebraic and Coalgebraic Methods in the Mathematics of Program Construction (R. Backhouse, R. L. Crole, and J. Gibbons, eds.), Lecture Notes in Computer Science, vol. 2297, Springer, 2002, pp. 89–150.

[BB97] J. C. M. Baeten and J. A. Bergstra, *Process algebra with propositional signals*, ACP '95: Algebra of Communicating Processes, Elsevier, 1997, pp. 381–405.

[BD02] M. Broy and E. Denert, *Software pioneers: Contributions to software engineering*, Springer, 2002.

[BD06] C. Bolduc and J. Desharnais, *Static analysis of programs using omega algebra with tests*, Relational Methods in Computer Science (W. MacCaull, M. Winter, and I. Düntsch, eds.), Lecture Notes in Computer Science, vol. 3929, Springer, 2006, pp. 60–72.

[BÉ93] S. L. Bloom and Z. Ésik, *Equational axioms for regular sets*, Mathematical Structures in Computer Science **3** (1993), no. 1, 1–24.

[BG98] L. Bachmair and H. Ganzinger, *Ordered chaining calculi for first-order theories of transitive relations*, Journal of the ACM **45** (1998), no. 6, 1007–1049.

[BGM93] A. Back, J. Guckenheimer, and M. Myers, *A dynamical simulation facility for hybrid systems*, Hybrid Systems (R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, eds.), Lecture Notes in Computer Science, vol. 736, Springer, 1993, pp. 255–267.

[BGMS07] D. Bresolin, V. Goranko, A. Montanari, and G. Sciavicco, *On decidability and expressiveness of propositional interval neighborhood logics*, Logical Foundations of Computer Science (S. N. Artëmov and A. Nerode, eds.), Lecture Notes in Computer Science, vol. 4514, Springer, 2007, pp. 84–99.

[BHF96] A. Buch, T. Hillenbrand, and R. Fettig, *Waldmeister: High Performance Equational Theorem Proving*, Proceedings of the International Symposium on Design and Implementation of Symbolic Computation Systems (J. Calmet and C. Limongelli, eds.), Lecture Notes in Computer Science, no. 1128, Springer, 1996, pp. 63–64.

[Bir67] G. Birkhoff, *Lattice theory*, 3rd ed., American Mathematical Society, 1967.

[BKS85] W. W. Bledsoe, K. Kunen, and R. Shostak, *Completeness results for inequality provers*, Artificial Intelligence **27** (1985), no. 3, 255–288.

[BM02] J. C. M. Baeten and C. A. Middelburg, *Process algebra with timing*, Monographs in Theoretical Computer Science, Springer, 2002.

[BM04] R. Backhouse and D. Michaelis, *Fixed-point characterisation of winning strategies in impartial games*, Relational and Kleene-Algebraic Methods in Computer Science (R. Berghammer, B. Möller, and G. Struth, eds.), Lecture Notes in Computer Science, vol. 3051, Springer, 2004, pp. 34–47.

[BM05] J. A. Bergstra and C. A. Middleburg, *Process algebra for hybrid systems*, Theoretical Computer Science **335** (2005), no. 2-3, 215–280.

[Bro93] R. W. Brockett, *Hybrid models for motion control*, Essays in Control: Perspectives in the Theory and its Applications (H. L. Trentelman and J.C. Willems, eds.), Birkhäuser, 1993, pp. 29–53.

[BRZ00] R. Barua, S. Roy, and C. Zhou, *Completeness of Neighbourhood Logic*, Journal of Logic and Computation **10** (2000), no. 2, 271–295.

[BS81] S. Burris and H. P. Sankappanavar, *A course in universal algebra (graduate texts in mathematics)*, 2nd ed., Springer, 1981, (The Millennium Edition) available at `http://www.math.uwaterloo.ca/~snburris/htdocs/ualg.html`.

[BS98] S Bornot and J. Sifakis, *On the composition of hybrid systems*, HSCC '98: Proceedings of the First International Workshop on Hybrid Systems, Lecture Notes in Computer Science, vol. 1386, Springer, 1998, pp. 49–63.

[BS01] M. Broy and K. Stølen, *Specification and development of interactive systems: Focus on streams, interfaces, and refinement*, Springer, 2001.

[BvW98] R.-J. Back and J. von Wright, *Refinement calculus: A systematic introduction*, Graduate Texts in Computer Science, Springer, 1998.

[BvW99] ———, *Reasoning algebraically about loops*, Acta Informatica **36** (1999), no. 4, 295–334.

[BW90] J. C. M. Baeten and W. P. Weijland, *Process algebra*, Cambridge University Press, 1990.

[CFP00] E. M. Clarke, O. Frumberg, and D. Peled, *Model checking*, MIT Press, 2000.

[CK03] A. Chutinan and B. H Krogh, *Computational techniques for hybrid system verification*, IEEE Transactions on Automatic Control **48** (2003), no. 1, 64–75.

[CL06] C. G. Cassandras and J. Lygeros (eds.), *Stochastic Hybrid Systems*, Control Engineering Series, vol. 24, CRC Press and IEEE Press, 2006.

[Coh94] E. Cohen, *Using Kleene algebra to reason about concurrency control*, Tech. report, Telcordia, 1994.

[Coh00] ———, *Separation and reduction*, Mathematics of Program Construction (MPC 2000) (R. Backhouse and J. N. Oliveira, eds.), Lecture Notes in Computer Science, vol. 1837, Springer, 2000, pp. 45–59.

[Con71] J. H. Conway, *Regular algebra and finite machines*, Chapman & Hall, 1971.

[Cor88] J. M. Corbett, *Designing hybrid automated manufacturing systems: A European perspective*, Proceedings of the First International Conference on Ergonomics of Hybrid Automated Systems I, Elsevier, 1988, pp. 167–172.

[Dav99] J. M. Davoren, *On hybrid systems and the modal μ-calculus*, Hybrid Systems V, Lecture Notes in Computer Science, vol. 1567, Springer, 1999, pp. 38–69.

[DH08] H.-H. Dang and P. Höfner, *First-order theorem prover evaluation w.r.t. Relation- and Kleene algebra*, Relations and Kleene Algebra in Computer Science — PhD Programme at RelMiCS 10/AKA 05 (R. Berghammer, B. Möller, and G. Struth, eds.), Technical Report, no. 2008-04, Institut für Informatik, Universität Augsburg, 2008, pp. 48–52.

[DHO04] W. Damm, H. Hungar, and E.-R. Olderog, *On the verification of cooperating traffic agents*, Formal Methods for Components and Objects (F. S. de Boer, M. M. Bonsangue, S. Graf, and W. P. de Roever, eds.), Lecture Notes in Computer Science, vol. 3188, Springer, 2004, pp. 77–110.

[Dij75] E. W. Dijkstra, *Guarded commands, non-determinacy and formal derivation of programs*, Communications of the ACM **18** (1975), no. 8, 453–457.

[Dij00] R. M. Dijkstra, *Computation calculus bridging a formalisation gap*, Science of Computer Programming **37** (2000), 3–36.

[DLL+06] S. Dimopoulos, I. Legouras, Z. Lygerou, G. Xouri, and J. Lygeros, *DNA replication in the fission yeast: Robustness in the face of uncertainty*, Yeast **23** (2006), 951–962.

[DM01] J. Desharnais and B. Möller, *Characterizing determinacy in Kleene algebra*, Information Sciences **139** (2001), 253–273.

[DMS04a] J. Desharnais, B. Möller, and G. Struth, *Modal Kleene algebra and applications — A survey*, Relational Methods in Computer Science **1** (2004), 93–131.

[DMS04b] ———, *Termination in modal Kleene algebra*, Exploring New Frontiers of Theoretical Informatics, 3rd International Conference on Theoretical Computer Science (TCS2004) (J.-J. Lévy, E. W. Mayr, and J. C. Mitchell, eds.), Kluwer, 2004, Revised version: *Algebraic Notions of Termination*. Technical Report 2006-23, Institut für Informatik, Universität Augsburg, 2006, pp. 647–660.

[DMS06] ———, *Kleene algebra with domain*, ACM Transactions on Computational Logic **7** (2006), no. 4, 798–833.

[DN00] J. M. Davoren and A. Nerode, *Logics for hybrid systems*, Proceedings of the IEEE **88** (2000), no. 7, 985–1010.

[DP02] B. A. Davey and H. A. Priestley, *Introduction to lattices and order*, 2nd ed., Cambridge University Press, 2002.

[DS08] J. Desharnais and G. Struth, *Modal Semirings Revisited*, MPC '08: Mathematics of Program Construction, Lecture Notes in Computer Science, vol. 5133, Springer, 2008, pp. 360–387.

[Dut95a] B. Dutertre, *Complete proof systems for first order interval temporal logic*, Tenth annual IEEE Symposium on logic in computer science (LICS), IEEE Press, 1995, pp. 36–43.

[Dut95b] ———, *On first-order interval temporal logic*, Report CSD-TR-94-3, Department of Computer Science, Royal Holloway, University of London, 1995.

[EC81] E. A. Emerson and E. M. Clarke, *Using branching time temporal logic to synthesize synchronization skeletons*, Science of Computer Programming **2** (1981), no. 3, 241–266.

[Ehm04] T. Ehm, *Pointer Kleene algebra*, Relational and Kleene-Algebraic Methods in Computer Science (R. Berghammer, B. Möller, and G. Struth, eds.), Lecture Notes in Computer Science, vol. 3051, Springer, 2004, pp. 99–111.

[Eme91] E. A. Emerson, *Formal models and semantics*, Handbook of Theoretical Computer Science (vol. B) (1991), 995–1072.

[FLP06] M. F. Frías and C. López Pombo, *Interpretability of linear time temporal logic in fork algebra*, Journal of Logic and Algebraic Programming **66** (2006), no. 2, 161–184.

[FM06] J. Faber and R. Meyer, *Model checking data-dependent real-time properties of the European train control system*, FMCAD '06: Proceedings of the Formal Methods in Computer Aided Design, IEEE Press, 2006, pp. 76–77.

[Frä99] M. Fränzle, *Analysis of hybrid systems: An ounce of realism can save an infinity of states*, 13th International Workshop and 8th Annual Conference of the EACSL on Computer Science Logic (CSL), Lecture Notes in Computer Science, vol. 1683, Springer, 1999, pp. 126–140.

[GM97] S. Gaubert and Max Plus, *Methods and applications of* $(max, +)$ *linear algebra*, STACS '97: Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 1200, Springer, 1997, pp. 261–282.

[GMS04] V. Goranko, A. Montanari, and G. Sciavicco, *A road map of interval temporal logics and duration calculi*, Journal of Applied Non-Classical Logics **14** (2004), no. 1-2, 9–54.

[GT04] R. Ghosh and C. Tomlin, *Symbolic reachable set computation of piecewise affine hybrid automata and its application to biological modeling: Delta-Notch protein signaling*, IEEE Transactions on Systems Biology **1** (2004), no. 1, 170–183.

[Har79] D. Harel, *First-order dynamic logic*, Springer, 1979.

[HBV97] T. Hillenbrand, A Buch, and R Vogt, *Waldmeister: High performance equational deduction*, Journal of Automated Reasoning **18** (1997), 265–270.

[HC96] G. E. Hughes and M. J. Cresswell, *A new introduction to modal logic*, Routledge, 1996.

[He94] J. He, *From CSP to hybrid systems*, A classical mind: Essays in honour of C.A.R. Hoare (A.W. Roscoe, ed.), Prentice Hall, 1994, pp. 171–189.

[Hen96] T. A. Henzinger, *The theory of hybrid automata*, IEEE Symposium on Logic in Computer Science, IEEE Press, 1996, pp. 278–292.

[Hen00] ———, *The theory of hybrid automata*, Verification of Digital and Hybrid Systems (M. K. Inan and M. K. Kurshan, eds.), NATO ASI Series F: Computer and Systems Sciences, vol. 170, Springer, 2000, (Extended version of [Hen96]), pp. 265–292.

[HGK98] P. Herrmann, G. Graw, and H. Krumm, *Compositional specification and structured verification of hybrid systems in cTLA*, ISORC '98: Proceedings of the The 1st IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, IEEE Press, 1998, pp. 335–350.

[HHM99] T. A. Henzinger, B. Horowitz, and R. Majumdar, *Rectangular hybrid games*, 10th International Conference on Concurrency Theory (CONCUR'99), Lecture Notes in Computer Science, Springer, 1999, pp. 320–335.

[HHWT97] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, *HYTECH: A model checker for hybrid systems*, CAV '97: Conference on Computer Aided Verification, Lecture Notes in Computer Science, vol. 1254, Springer, 1997, pp. 460–463.

[HK97] P. Herrmann and H. Krumm, *Specification of hybrid systems in cTLA+*, Parallel and Distributed Real-Time Systems, Workshop (1997), 212–216.

[HKT00] D. Harel, D. Kozen, and J. Tiuryn, *Dynamic logic*, MIT Press, 2000.

[HLP90] E. Harel, O. Lichtenstein, and A. Pnueli, *Explicit clock temporal logic*, Logic in Computer Science, 1990. LICS '90, Proceedings., Fifth Annual IEEE Symposium on e (1990), 402–413.

[HM00] T. A. Henzinger and R. Majumdar, *Symbolic model checking for rectangular hybrid systems*, TACAS '00, Lecture Notes in Computer Science, vol. 1770, Springer, 2000, pp. 142–156.

[HM06] P. Höfner and B. Möller, *Lazy semiring neighbours and some applications*, Relations and Kleene Algebra in Computer Science (R. A. Schmidt, ed.), Lecture Notes in Computer Science, vol. 4136, Springer, 2006, pp. 207–221.

[HM08] ———, *Algebraic neighbourhood logic*, Journal of Logic and Algebraic Programming **76** (2008), 35–59.

[HMM83] J. Y. Halpern, Z. Manna, and B. C. Moszkowski, *A hardware semantics based on temporal intervals*, ICALP Automata, Languages and Programming (J. Díaz, ed.), Lecture Notes in Computer Science, vol. 154, Springer, 1983, pp. 278–291.

[HMS06] P. Höfner, B. Möller, and K. Solin, *Omega algebra, demonic refinement algebra and commands*, Relations and Kleene Algebra in Computer Science (R. A. Schmidt, ed.), Lecture Notes in Computer Science, vol. 4136, Springer, 2006, pp. 222–234.

[HMSW09] C. A. R. Hoare, B. Möller, G. Struth, and I. Wehrman, *Concurrent Kleene algebra*, 20th International Conference on Concurrency Theory (CONCUR'09) (M. Bravetti and G. Zavattaro, eds.), Lecture Notes in Computer Science, vol. 5710, Springer, 2009, pp. 399–414.

[HMU07] J. E. Hopcroft, R. Motwani, and J. D. Ullmann, *Introduction to automata theory, languages, and computation*, 3 ed., Addison-Wesley, 2007.

[HNSY94] T. A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine, *Symbolic model checking for real-time systems*, Information and Computation **111** (1994), 394–406.

[Hoa69] C. A. R. Hoare, *An axiomatic basis for computer programming*, Communications of the ACM **12** (1969), no. 10, 576–580, Reprint in [BD02].

[Höf] P. Höfner, *Database for automated proofs of Kleene algebra*, `http://www.dcs.shef.ac.uk/~georg/ka` (accessed November 13, 2009).

[Höf03] ———, *Von sequentieller Algebra zu Kleene-algebra: Intervalloperatoren und Zeitdauerkalkül*, Master's thesis, Universität Augsburg, 2003, (In German).

[Höf05a] ———, *An algebraic semantics for Duration Calculus*, Proceedings of the 10th ESSLLI Student Session (J. Gervain, ed.), Heriot-Watt University, 2005, pp. 99–111.

[Höf05b] ———, *From sequential algebra to Kleene algebra: Interval modalities and Duration Calculus*, Tech. Report 2005-07, Institut für Informatik, Universität Augsburg, 2005.

[Höf06] ———, *f-generated Kleene algebra*, Relations and Kleene Algebra in Computer Science — PhD Programme at RelMiCS/AKA 2006 (R. A. Schmidt and G. Struth, eds.), Research Report, no. CS-06-09, Department of Computer Science, University of Sheffield, 2006, pp. 55–59.

[Höf07] ———, *Semiring neighbours: An algebraic embedding and extension of Neighbourhood Logic*, Electronic Notes in Theoretical Computer Science **191** (2007), 49–72.

[Höf08a] K. Höfner, *Personal communication*, 2008.

[Höf08b] P. Höfner, *Automated reasoning for hybrid systems — Two case studies*, Relations and Kleene Algebra in Computer Science (R. Berghammer, B. Möller, and G. Struth, eds.), Lecture Notes in Computer Science, vol. 4988, Springer, 2008, pp. 191–205.

[Hol98] M. Hollenberg, *Equational axioms of test algebra*, CSL '97: Selected Papers from the 11th International Workshop on Computer Science Logic (M. Nielsen and W. Thomas, eds.), Lecture Notes in Computer Science, vol. 1414, 1998.

[HS91] J. Y. Halpern and Y. Shoham, *A propositional modal logic of time intervals*, Journal of the ACM **38** (1991), no. 4, 935–962.

[HS07] P. Höfner and G. Struth, *Automated reasoning in Kleene algebra*, Automated Deduction — CADE-21 (F. Pfennig, ed.), Lecture Notes in Artificial Intelligence, vol. 4603, Springer, 2007, pp. 279–294.

[HS08a] ———, *Can refinement be automated?*, Refine 2007 (E. Boiten, J. Derrick, and G. Smith, eds.), vol. 201, Electronic Notes in Theoretical Computer Science, no. 197-222, Elsevier, 2008.

[HS08b] ———, *Non-termination in idempotent semirings*, Relations and Kleene Algebra in Computer Science (R. Berghammer, B. Möller, and G. Struth, eds.), Lecture Notes in Computer Science, vol. 4988, Springer, 2008, pp. 206–220.

[HS08c] ———, *On automating the calculus of relations*, Automated Reasoning (IJCAR 2008) (A. Armando, P. Baumgartner, and G. Dowek, eds.), Lecture Notes in Computer Science, vol. 5159, Springer, 2008, pp. 50–66.

[HSS09] P. Höfner, G. Struth, and G. Sutcliffe, *Automated verification of refinement laws*, Annals of Mathematics and Artificial Intelligence, Special Issue on First-order Theorem Proving **35** (2009), no. 1, 35–62.

[HW98] U. Hebisch and H. J. Weinert, *Semirings — Algebraic theory and applications in computer science*, World Scientific, 1998.

[HYS] HYSCOM, *IEEE Control System Society on hybrid systems*, `http://www.dii.unisi.it/hybrid/ieee/`, (accessed February 15, 2008).

[Isa65] R. Isaacs, *Differential games*, Wiley, 1965, Republished: Dover, 1999.

[JELS99] K. H. Johansson, M. Egerstedt, J. Lygeros, and Sastry. S., *On the regularization of Zeno hybrid automata*, Systems & Control Letters **38** (1999), 141–150.

[Kah04] W. Kahl, *Calculational relation-algebraic proofs in Isabelle/Isar*, Relational and Kleene-Algebraic Methods in Computer Science (R. Berghammer, B. Möller, and G. Struth, eds.), Lecture Notes in Computer Science, vol. 3051, Springer, 2004, pp. 179–190.

[Kle51] S. C. Kleene, *Representation of events in nerve nets and finite automata*, Tech. Report RM-704, RAND Corporation, 1951, RAND Research Memorandum.

[Kle56] ———, *Representation of events in nerve nets and finite automata*, Automata Studies (C.E. Shannon and J. McCarthy, eds.), Annals of Mathematics Studies, vol. 34, Princeton University Press, 1956, pp. 3–41.

[Koz94] D. Kozen, *A completeness theorem for Kleene algebras and the algebra of regular events*, Information and Computation **110** (1994), no. 2, 366–390.

[Koz97] ———, *Kleene algebra with tests*, ACM Transactions on Programming Languages and Systems **19** (1997), no. 3, 427–443.

[Koz00] ———, *On Hoare logic and Kleene algebra with tests*, ACM Transactions on Computational Logic **1** (2000), no. 1, 60–76.

[Koz03] ———, *Automata on guarded strings and applications*, Matématica Contemporânea **24** (2003), 117–139.

[Kro91] D. Krob, *Complete systems of b-rational identities*, Theoretical Computer Science **89** (1991), no. 2, 207–343.

[KS86] W. Kuich and A. Salomaa, *Semirings, automata, languages*, Springer, 1986.

[Lam77] L. Lamport, *Proving the correctness of multiprocess programs*, IEEE Transactions on Software Engineering **3** (1977), no. 2, 125–143.

[Lam93] ———, *Hybrid systems in TLA$^+$*, Hybrid Systems (R. Grossman, A. Nerode, and A.P. Ravn, eds.), Lecture Notes in Computer Science, vol. 736, Springer, 1993, pp. 77–102.

[Lam02] ———, *Specifying systems: The TLA$^+$ language and tools for hardware and software engineers*, Addison-Wesley, 2002.

[LJS+03] J. Lygeros, K. H. Johansson, S. N. Simić, J. Zhang, and Sastry S. S., *Dynamical properties of hybrid automata*, IEEE Transactions on Automatic Control **48** (2003), 2–17.

[LSV03] N. A. Lynch, R. Segala, and F. W. Vaandrager, *Hybrid I/O automata*, Information and Computation **185** (2003), no. 1, 105–157.

[LT04] P. Lincoln and A. Tiwari, *Symbolic systems biology: Hybrid modeling and analysis of biological networks*, Hybrid Systems: Computation and Control (R. Alur and G. Pappas, eds.), Lecture Notes in Computer Science, vol. 2993, Springer, 2004, pp. 660–672.

[Lyg04] J. Lygeros, *Lecture notes on hybrid systems*, 2004.

[Mad06] R. Maddux, *Relation algebras*, Studies in Logic and the Foundations of Mathematics, vol. 150, Elsevier, 2006.

[MB85] E. Manes and D. Benson, *The inverse semigroup of a sum-ordered semiring*, Semigroup Forum **31** (1985), 129–152.

[McC] W. W. McCune, *Prover9 and Mace4*, `http://www.cs.unm.edu/~mccune/prover9` (accessed November 13, 2009).

[MGCM08] A. K. McIver, C. Gonzalia, E. Cohen, and C. C. Morgan, *Using probabilistic Kleene algebra pKA for protocol verification*, Journal of Logic and Algebraic Programming **76** (2008), no. 1, 90–111.

[MH08] L. Meinicke and I. Hayes, *Algebraic reasoning for probabilistic action systems and while-loops*, Acta Informatica **45** (2008), no. 5, 321–382.

[MHS06] B. Möller, P. Höfner, and G. Struth, *Quantales and temporal logics*, Algebraic Methodology and Software Technology (M. Johnson and V. Vene, eds.), Lecture Notes in Computer Science, vol. 4019, Springer, 2006, pp. 263–277.

[Mil99] R. Milner, *Communicating and mobile systems: The ϕ-calculus*, Cambridge University Press, 1999.

[Mil00] J. S. Miller, *Decidability and complexity results for timed automata and semi-linear hybrid automata*, Third International Workshop on Hybrid Systems: Computation and Control (HSCC 00), Lecture Notes in Computer Science, vol. 1790, Springer, 2000, pp. 296–309.

[MO02] W. MacCaull and E. Orłowska, *Correspondence results for relational proof systems with application to the lambek calculus*, Studia Logica **71** (2002), no. 3, 389–414.

[Möl97] B. Möller, *Calculating with pointer structures*, Proceedings of the IFIP TC2/WG 2.1 International Workshop on Algorithmic Languages and Calculi, Chapman & Hall, 1997, pp. 24–48.

[Möl04] ———, *Lazy Kleene algebra*, Mathematics of Program Construction (D. Kozen, ed.), Lecture Notes in Computer Science, vol. 3125, Springer, 2004, pp. 252–273.

[Möl05a] ———, *Complete tests do not guarantee domain*, Tech. Report 2005-6, Institut für Informatik, Universität Augsburg, 2005.

[Möl05b] ———, *Residuals and detachments*, Tech. Report 2005-20, Institut für Informatik, Universität Augsburg, 2005.

[Möl07] ———, *Kleene getting lazy*, Science of Computer Programming **65** (2007), 195–214.

[Mos00] B. C. Moszkowski, *A complete axiomatization of interval temporal logic with infinite time*, 15th Annual IEEE Symposium on Logic in Computer Science, IEEE Press, 2000, pp. 241–252.

[MP93a] Z. Manna and A. Pnueli, *Models for reactivity*, Acta Informatica **30** (1993), no. 7, 609–678.

[MP93b] ———, *Verifying hybrid systems*, Hybrid Systems (R. Grossman, A. Nerode, and A.P. Ravn, eds.), Lecture Notes in Computer Science, vol. 736, Springer, 1993, pp. 4–35.

[MS75] Z. Manna and A. Shamir, *The optimal fixedpoint of recursive programs*, STOC '75: Proceedings of seventh annual ACM symposium on Theory of computing, ACM Press, 1975, pp. 194–206.

[MS76] ———, *The theoretical aspects of the optimal fixed point*, SIAM Journal on Computing **5** (1976), no. 3, 414–426.

[MS95] Z. Manna and the STeP group, *STeP: The Stanford temporal prover (Educational release), User's manual*, Tech. Report STAN-CS-TR-95-1562, Computer Science Department, Stanford University, 1995.

[MS98] Z. Manna and H. Sipma, *Deductive verification of hybrid systems using STeP*, HSCC '98: Proceedings of the First International Workshop on Hybrid Systems, Lecture Notes in Computer Science, Springer, 1998, pp. 305–318.

[MS06] B. Möller and G. Struth, *Algebras of modal operators and partial correctness*, Theoretical Computer Science **351** (2006), no. 2, 221–239.

[MS08] L. Meinicke and K. Solin, *Refinement algebra for probabilistic programs*, Electronic Notes in Theoretical Computer Science **201** (2008), 177–195.

[Nie] J. Niehaus, *AVACS Homepage*, `http://www.avacs.org` (accessed September 3, 2008).

[Nig95] E. Nigg, *Cyclin-dependent protein kinases: Key regulators of the eukaryotic cell cycle*, BioEssays **17** (1995), no. 6, 471 – 480.

[ORS92] S. Owre, J. M. Rushby, and N. Shankar, *PVS: A prototype verification system*, 11th International Conference on Automated Deduction (CADE) (D. Kapur, ed.), Lecture Notes in Artificial Intelligence, vol. 607, Springer, 1992, pp. 748–752.

[Ost89a] J. S. Ostroff, *Synthesis of controllers for real-time discrete event systems*, IEEE Proceedings of the 28th Conference on Decision and Control, IEEE Press, 1989.

[Ost89b] ———, *Temporal logic for real-time systems*, Advanced Software Development Series, Wiley, 1989.

[Pan96] P. K. Pandya, *Weak chop inverses and liveness in mean-value calculus*, Formal Techniques in Real-Time and Fault-Tolerant Systems (B. Jonsson and J. Parrow, eds.), Lecture Notes in Computer Science, vol. 1135, Springer, 1996, pp. 148–167.

[Par80] D. Park, *On the semantics of fair parallelism*, Proceedings of the Abstract Software Specifications, 1979 Copenhagen Winter School (D. Bjørner, ed.), Lecture Notes in Computer Science, Springer, 1980, pp. 504–526.

[Pla08] A. Platzer, *Differential dynamic logic for hybrid systems.*, Journal of Automated Reasoning **41** (2008), no. 2, 143–189.

[PQ08] A. Platzer and J.-D. Quesel, *KeYmaera: A hybrid theorem prover for hybrid systems.*, Automated Reasoning (A. Armando, P. Baumgartner, and G. Dowek, eds.), Lecture Notes in Artificial Intelligence, vol. 5195, Springer, 2008, pp. 171–178.

[Pra92] V. R. Pratt, *Dynamic algebras: Examples, constructions, applications*, Studia Logica **50** (1992), no. 3/4, 571–605.

[Puj97] A. K. Pujari, *Neighbourhood logic and interval algebra*, Tech. Report 116, UNU/IIST, 1997.

[PW06] V. Prevesto and U. Waldmann, *SPASS+T*, ESCoR: Empirically Successful Computerized Reasoning (G. Sutcliffe, R. Schmidt, and S. Schulz, eds.), CEUR Workshop Proceedings, vol. 192, 2006, pp. 18–33.

[Pyz03] T. Pyzdek, *Quality engineering handbook*, 2nd ed., Marcel Dekker Inc, 2003.

[Rab00] A. M. Rabinovich, *Expressive completeness of duration calculus.*, Information and Computation **156** (2000), no. 1-2, 320–344.

[Red64] V. N. Redko, *On defining relations for the algebra of regular events*, Ukrainskii Matematicheskii Zhurnal **16** (1964), 120–126, (In Russian).

[Rey01] M. Reynolds, *An axiomatization of full computation tree logic*, Journal of Symbolic Logic **66** (2001), no. 3, 1011–1057.

[Rey05] ———, *An axiomatization of PCTL*.*, Information and Computation **201** (2005), no. 1, 72–119.

[Rön01] M. Rönkkö, *Stepwise development of hybrid systems*, Ph.D. thesis, Department of Computer Science, Åbo Akademi University, 2001.

[RS97] G. Rozenberg and A. Salomaa (eds.), *Handbook of formal languages, vol. 1: Word, language, grammar*, Springer, 1997.

[RS03] W. C. Rounds and H. Song, *The ϕ-calculus: A language for distributed control of reconfigurable embedded systems*, Hybrid Systems: Computation and Control (O. Maler and A. Pnueli, eds.), Lecture Notes in Computer Science, vol. 2623, Springer, 2003, pp. 435–449.

[RV01] J. A. Robinson and A. Voronkov, *Handbook of automated reasoning (in 2 volumes)*, Elsevier and MIT Press, 2001.

[RW87] P. J. Ramadge and W. M. Wonham, *Modular feedback logic for discrete event systems*, SIAM Journal on Control and Optimization **25** (1987), no. 5, 1202–1218.

[RZ97] S. Roy and C. Zhou, *Notes on neighbourhood logic*, Tech. Report 97, The United Nations University UNU/IIST, 1997.

[SAFM06] R. St-Aubin, J. Friedman, and A. K. Mackworth, *A formal mathematical framework for modeling probabilistic hybrid systems*, Annals of Mathematics and Artificial Intelligence **47** (2006), no. 3-4, 397–425.

[Sal66] A. Salomaa, *Two complete axiom systems for the algebra of regular events*, Journal of the ACM **13** (1966), no. 1, 158–169.

[SDJ+92] S. Schneider, J. Davies, D. M. Jackson, G. M. Reed, J. N. Reed, and A. W. Roscoe, *Timed CSP: Theory and practice*, Proceedings of the Real-Time: Theory in Practice, REX Workshop, Springer, 1992, pp. 640–675.

[Sin04] M. Sintzoff, *Iterative synthesis of control guards ensuring invariance and inevitability in discrete-decision games*, From Object-Orientation to Formal Methods, Essays in Memory of Ole-Johan Dahl (O. Owe, S. Krogdahl, and T. Lyche, eds.), Lecture Notes in Computer Science, vol. 2635, Springer, 2004.

[Ska94] J. U. Skakkebæk, *Liveness and fairness in duration calculus*, 5th International Conference on Concurrency Theory (CONCUR'94) (B. Jonsson and J. Parrow, eds.), Lecture Notes in Computer Science, vol. 836, Springer, 1994, pp. 283–298.

[SM56] C. E. Shannon and J. McCarthy, *Automata Studies. (AM-34) (Annals of Mathematics Studies)*, Princeton University Press, 1956.

[SMT+95] S. Sastry, G. Meyer, C. Tomlin, J. Lygeros, D. Godbole, and G. Pappas, *Hybrid control in air traffic management systems*, IEEE Conference in Decision and Control (vol. 2), IEEE Press, 1995, pp. 1478 – 1483.

[SN96] B. Stern and P. Nurse, *A quantitative model for the cdc2 control of s phase and mitosis in fission yeast*, Trends in Genetics **12** (1996), no. 9, 345 – 350.

[SP07] G. Sutcliffe and Y. Puzis, *SRASS—A semantic relevance axiom selection system*, Automated deduction — CADE-21 (F. Pfennig, ed.), Lecture Notes in Artificial Intelligence, vol. 4603, Springer, 2007, pp. 295–310.

[Spe07] *Spektrum der Wissenschaft Spezial*, vol. 3/07, 2007, (In German).

[SPS00] O. Shakernia, G.J. Pappas, and S. Sastry, *Semidecidable controller synthesis for classes of linear hybrid systems*, Decision and Control **2** (2000), 1834–1839.

[SS93] G. Schmidt and T. Ströhlein, *Relations and graphs: Discrete mathematics for computer scientists*, Springer, 1993.

[SS98] G. Sutcliffe and C. B. Suttner, *The TPTP problem library: CNF release v1.2.1*, Journal of Automated Reasoning **21** (1998), no. 2, 177–203.

[Sta97] L. Staiger, *Omega languages*, Handbook of Formal Languages **3** (1997), 339–387.

[Str02] G. Struth, *Calculating Church-Rosser proofs in Kleene algebra*, Relational Methods in Computer Science (H. de Swart, ed.), Lecture Notes in Computer Science, vol. 2561, Springer, 2002, pp. 276–290.

[Str06] ———, *Abstract abstract reduction.*, Journal of Logic and Algebraic Programming **66** (2006), no. 2, 239–270.

[Str07] ———, *Reasoning automatically about termination and refinement*, 6th International Workshop on First-Order Theorem Proving (S. Ranise, ed.), vol. Technical Report ULCS-07-018, Department of Computer Science, University of Liverpool, 2007, pp. 36–51.

[Tar41] A. Tarski, *On the calculus of relations*, Journal of Symbolic Logic **6** (1941), no. 3, 73–89.

[Ven90] Y. Venema, *Expressiveness and completeness of an interval tense logic.*, Notre Dame Journal of Formal Logic **31** (1990), no. 4, 529–547.

[Ven91] ———, *A modal logic for chopping intervals*, Journal of Logic and Computation **1** (1991), no. 4, 453–476.

[vK98] B. von Karger, *Temporal algebra*, Mathematical Structures in Computer Science **8** (1998), no. 3, 277–320.

[vKB98] B. von Karger and R. Berghammer, *A relational model for temporal logic*, Logic Journal of the IGPL **6** (1998), 157–173.

[vM99] M. von Mohrenschildt, *Closed form solutions of hybrid systems*, CRL Report 371, Faculty of Engineering, McMaster University, 1999.

[vN66] J. von Neumann, *Theory of self-reproducing automata*, University of Illinois Press, 1966.

[vOG97] D. von Oheimb and T. F. Gritzner, *Rall: Machine-supported proofs for relation algebra*, CADE 1994 (W. W. McCune, ed.), Lecture Notes in Computer Science, vol. 1249, Springer, 1997, pp. 380–394.

[vW02] J. von Wright, *From Kleene algebra to refinement algebra*, Mathematics of Program Construction (E. A. Boiten and B. Möller, eds.), Lecture Notes in Computer Science, vol. 2386, Springer, 2002, pp. 233–262.

[vW04] ———, *Towards a refinement algebra*, Science of Computer Programming **51** (2004), no. 1-2, 23–45.

[WSH+07] C. Weidenbach, R. Schmidt, T. Hillenbrand, R. Rusev, and D. Topic, *SPASS Version 3.0*, Automated deduction — CADE-21 (F. Pfenning, ed.), Lecture Notes in Artificial Intelligence, vol. 4603, Springer, 2007, pp. 514–520.

[WX04] H. Wang and Q. Xu, *Completeness of temporal logics over infinite intervals*, Discrete Applied Mathematics **136** (2004), no. 1, 87–103.

[ZH98] C. Zhou and M. R. Hansen, *An adequate first order interval logic*, Compositionality: The Significant Difference, International Symposium, (COMPOS 97) (W. P. de Roever, H. Langmaack, and A. Pnueli, eds.), Lecture Notes in Computer Science, vol. 1536, Springer, 1998, pp. 584–608.

[ZH04] ———, *Duration calculus: A formal approach to real-time systems*, Monographs in Theoretical Computer Science. An EATCS Series, Springer, 2004.

[ZHR91] C. Zhou, C. A. R. Hoare, and A. P. Ravn, *A calculus of durations*, Information Processing Letters **40** (1991), no. 5, 269–276.

[ZRH93] C. Zhou, A. P. Ravn, and M. R. Hansen, *An extended duration calculus for hybrid real-time systems*, Hybrid Systems (R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, eds.), Lecture Notes in Computer Science, vol. 736, Springer, 1993, pp. 36–59.

[Zus67] K. Zuse, *Rechnender Raum*, Elektronische Datenverarbeitung **8** (1967), 336–344, Reprint in [Spe07].

[ZVHX95] C. Zhou, D. Van Hung, and L. Xiaoshan, *A duration calculus with infinite intervals*, Fundamentals of Computation Theory, Springer, 1995, pp. 16–41.

List of Figures

2.1 Thermostat automaton 16
2.2 An accepted run of the temperature controller 17
2.3 Extended thermostat automaton 18
2.4 Hybrid automaton of a bouncing ball and corresponding run w.r.t. x_1 21
2.5 Hybrid automaton of a car with four gears 22
2.6 Cell-devision cycle and trajectories for CDK 23
2.7 Strict parallel composition of two simple hybrid automata 27
2.8 Liberal parallel composition of two simple hybrid automata 29
2.9 Architecture of a simple railroad system 30
2.10 Train automaton 31
2.11 Gate automaton 31
2.12 Controller automaton 32
2.13 Architecture of a railway system with a shared single-track way 33
2.14 Another train automaton 34
2.15 Strict parallel composed automaton for a shared single-track way . . . 35

3.1 Right-distributivity does not hold in general 41

4.1 Composition of two finite trajectories 63
4.2 Statements of hybrid programs 69
4.3 Constructed finite-state machine 73
4.4 State elimination technique by an example 74
4.5 Modified bouncing ball 85

4.6 Local linearity 91
4.7 Composed trajectories satisfying $\mathbb{E}p \cdot \Box q$ 92
4.8 Different types of trajectories of $(\diamondsuit P)^\dagger$ 95
4.9 Specification of a gearbox 97

5.1 Left and right neighbouhoods of an interval $[y, z]$ 112
5.2 Nested neigbourhood modalities 116

6.1 A simple system for route planning 131
6.2 An alternative route planning automaton 132
6.3 An assembly line scheduler 136

C.1 Strict parallel composed automaton for the gate controller 176
C.1 Strict parallel composed automaton for the gate controller (continued) 177
C.2 Liberal parallel composed automaton for the gate controller 178
C.2 Liberal parallel composed automaton for the gate controller (continued) 179
C.3 Strict parallel composed automaton for a shared single-track way . . . 180
C.4 Liberal parallel composed automaton for a shared single-track way . . 181

Index

atomic propositions, 104

codomain, 57
compatibility relation, 66
composition relation, 25
computation tree logic, *see* CTL*
continuations
    existential, 98
    universal, 98
control graph, 19
CTL*, 104
    algebraic semantics, 106
    formula, 104
    path formula, 105
    semantics, 105
    state formula, 105

detachment, 90
domain, 55

extended supremum, 81

fixpoint fusion, 45
flow condition, 19

Galois connection, 56, 90, 93, 121

hybrid automaton, 18
    dimension, 18
    liberal parallel composition, 27
    product, 24
    strict parallel composition, 25
hybrid program, 69
hybrid system, **15**

i-semiring, *see* semiring
initial condition, 19
interval, 62
invariant condition, 19
iteration
    finite, 39
    infinite, 39
    iteration (for processes), 82

jump, 19
    jump condition, 19
    proper, 19

Kleene algebra, 38, **39**
    automation, 47
    lazy, 42
    weak, 42

liveness, 17, **85**
local linearity, 90

mode
    control mode, 19
    mode trace, *see* trace
modularity, 85

natural order, 38
neighbourhood logic, 110
    chop operator, 112

formula, 111
global function, 110
global relation, 111
global variable, 110
semantics, 111
temporal proposition letter, 111
temporal variable, 110
term, 111
vocabulary, 110
neighbourhoodlogic
algebraic semantics, 114

omega algebra, 39
lazy, 42
weak, 42

prefix relation, 79
process, 64
composition, 64
extended, 67
purely finite, 51
purely infinite, 51

quantale, 38
lazy, 42
weak, 42

range, 93
relation
partial function, 108
residual, 90
left, 90
run, 20

safety, 17, **85**
safety-close, *see* safety-closed semiring
semiring, 38
Boolean, 41
boundary, **118**
full, 38
idempotent, 38
lazy, 41
neighbour, 118, **118**
perfect boundary, 119
perfect neighbour, 119
product, 73
progressive, 86
safety-close, 89
separated lazy, 51
weak, 41
shunting rule, 42
specification, 96
submodularity, 85
switch, 19
synchronisable, 25

test, 53
shunting, 54
time domain, 61
trace
mode trace, 20
transition trace, 20
trajectory, 20, **62**
composition, 63
duration, 62
extended, 67
parallel composition, 75, 76
range, 62
transition, 19
transition trace, *see* trace

Zeno effect, 77
Zeno elements, 81
Zeno-free elements, 81

Curriculum Vitae

Peter Höfner
peter@hoefner-online.de

Education

Jul 2009	Doctorate, University of Augsburg, Germany
Oct 2003	Diploma in Mathematics (major) and in Computer Science (minor)
Jul 1997	Abitur (similar to "A" level)

Academic Positions

since Jul 2007	Researcher, University of Augsburg, Germany
Sep 2006 – Jul 2007	Visiting Scholar and Teaching Assistent, University of Sheffield, UK
Oct 2003 – Aug 2006	Researcher, University of Augsburg, Germany

Non-Academic Positions

Aug 2001 – Sep 2001	Research trainee Max-Planck-Institute for Plasma Physics, Garching near Munich, Germany
Oct 2000 – Jul 2003	Demonstrator, University Augsburg, Germany

Grants and Awards

Dec 2008	Award for young researchers, Universität Bayern e.V. (2nd place, interdisciplinary)
since Jul 2007	Research Scholarship by Universität Bayern e.V. (integrated in the Elite Network of Bavaria)
Aug 2008	Woody Bledsoe Travel Award 2008
Jul 2007	Woody Bledsoe Travel Award 2007
Aug 2005	Travel Grant by DFG